INDEX

Sr. No.	Topic	Page No.
1.	JavaScript Introduction	1
2.	JavaScript Basics	
	. Js Comment	6
	. Js Variable	7
	. Js Global Variable	8
	. Js Data Types	8
	. Js Operators	9
	. Js If Statements	12
	. Js Switch	14
	. Js Loop	15
	. Js Function	16
3.	JavaScript Objects	
	. Js Object	18
	. Js Array	22
	. Js String	24
	. Js Date	26
	. Js Math	28
	. Js Number	31
	. Js Boolean	32
4.	JavaScript BOM	
	Browser Objects	33
5.	JavaScript DOM	
	. Document Object	39

6. JavaScript Validation 45

7. JavaScript OOPs
- JS Class 47
- JS Object 49
- JS Prototype 54
- JS Constructor Method 55
- JS static Method 57
- JS Encapsulation 58
- JS Inheritance 60
- JS Polymorphism 61
- JS Abstraction 62

8. JavaScript Cookies 63

9. JavaScript Events 70

10. Exception Handling 88

11. JavaScript Misc
- JS this Keyword 93
- JS Debugging 95
- JS Hoisting 96
- JS Strict Mode 97
- JavaScript Promise 98
- JS Compare dates 102
- JavaScript array.length 105
- JavaScript alert() 106
- Javascript eval() function 107
- JavaScript closest() 108
- Javascript continue statement 109
- JS getAttribute()method 110

JavaScript

JavaScript is used to create client-side dynamic pages. JavaScript is an object-based scripting language which is lightweight and cross-platform. JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as Couch DB and Mongo DB uses JavaScript as their scripting and query language.

Features of JavaScript

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.

3. JavaScript is a weakly typed language, where certain types are implicitly cast.
4. JavaScript is an object-oriented programming language, wherever it is also use as prototypes rather than using classes for inheritance.
5. It is a lightweighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

History of JavaScript

In 1993, Mosaic, the first popular web browser, came into existence. In the year 1994, Netscape was founded by Marc Andreessen. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTMT to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited Brendan Eich intending to implement and embed scheme programming language to the browser. But before Brendan could start, the company merged with Sun Microsystems for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'JavaScript'.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

1. Client - Side Validation.
2. Dynamic drop-down menus.
3. Displaying date and time.
4. Displaying pop-up windows and dialog boxes.
5. Displaying clocks etc.

JavaScript Example

JavaScript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Example

```
<script type = "text/javascript">
document. Write ("JavaScript is a simple language");
</script>
```

The Script tag specifies that we are using JavaScript. The text/javascript is the content type that provides information to the browser about the data.
The document.Write() function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

3 Places to put JavaScript Code

1. JavaScript Example: Code between the body tag
In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type = "text/javascript">
alert ("Hello Javapoint");
</script>
```

2. JavaScript Example : Code between the head tag
We are creating a function msg(). To create function in JavaScript, you need to write function with function name as given below.
To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type = "text/javascript">
function msg(){
alert("Hello Javapoint");
}
</script>
</head>
<body>
<p> Welcome to Java Script </p>
<form>
<input type = "button" value = "click" onclick="msg()"/>
</form>
</body>
</html>
```

3. External JavaScript file :
We can create external JavaScript file and embed it in many html page.
It provides code re usability because single JavaScript file can be used in several html pages.

At external JavaScript file must be saved by js extension. It is recommended to embed all JavaScript files into a single file. It increase the speed of the webpage.
Let's create an external JavaScript
message.js

```
function msg(){
alert("Hello Javapoint");
}
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.
index.html

```
<html>
<head>
<script type="text/javascript" SrC="message.js"></script>
</head>
<body>
<p> Welcome to JavaScript</p>
<form>
<input type = "button" value = "Click" Onclick = msg()"/>
</form>
</body>
</html>
```

Advantages of External JavaScript

1. It helps in the resuability of code in more than one HTML file.
2. It allows easy code ra readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.

4. It enables both Web designers and coders to work with html and js files parallely and separately, ie., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The Web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

JavaScript Comment

The JavaScript comments are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions. so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript Comments

There are mainly two advantages of JS comments.

1. To make code easy to understand.
2. To avoid the unnecessary code.

Types of JavaScript Comments

1. Single - line Comment
2. Multi - line Comment

JavaScript Single line Comment
It is represented by double forward slashes (//).
It can be used before and after the statement.

```
<script>
//It is single line comment
document.Write ("hello javascript");
</script>
```

JavaScript Multi line Comment
It can be used to add single as well as multi line comments. So, it is more convenient.

```
/*your code here*/
```

JavaScript Variable

A JavaScript Variable is simply a name of storage location. There are two types of variables in JS: local variable and global variable.

Rules for declaring variable

1. Name must start with a letter (a to z or A to Z), underscore (_), or dollar ($) sign.
2. After first letter we can use digits (0 to 9), for example value 1.
3. JavaScript variables are case sensitive.

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only.

```
<script>
function abc (){
var x = 10;// local variable
}
</script>
```

JavaScript global Variable

A JavaScript global variable is accessible from any function. A variable i.e. declared outside the function or declared with window object is know as global variable.

```
<script>
var data = 200; // global variable
function a(){
}
</script>
```

JavaScript Data Types

JavaScript provides different data types to hold different types of values. There are two types of data types in JavaScript.

1. Primitive data type
2. Non-primitive data type

```
var a = 40;//holding number
var b = "Rahul";// holding string
```

JavaScript Primitive data types

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

JavaScript non-primitive data types

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Operators

JavaScript operators are symbols that are used to perform operations on operands.

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands.

Operator	Description	Example
+	Addition	10 + 20 = 30
−	Subtraction	20 − 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	20 % 10 = 0
++	Increment	var a = 10; a++; Now a = 11
−−	Decrement	var a = 10; a−−; Now a = 9

JavaScript Comparison Operators

The JavaScript comparison operator compares the 2 operands.

Operator	Description	Example
==	Is equal to	10==20 = false
===	Indentical	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

JavaScript Bitwise Operators

The bitwise operators perform bitwise operations on operands

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33)=false
|	Bitwise OR	(10==20 | 20==33)=false
^	Bitwise XOR	(10==20 ^ 20==33)=false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift 0	(10>>>2)=2

JavaScript Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

JavaScript Assignment Operators

The following operators are known as JavaScript assignment operators

Operator	Description	Example
=	Assign	10 + 10 = 20
+ =	Add and assign	Var a = 10; a+ = 20; Now a = 30
- =	Subtract and assign	Var a = 20; a- = 10; Now a = 10
* =	Multiply and assign	Var a = 10; a* = 20; Now a = 200
/ =	Divide and assign	Var a = 10; a/ = 2; Now a = 5
% =	Modulus and assign	Var a = 10; a % = 2; Now a = 0

JavaScript Special Operators

The following operators are known as Javascript special operators.

Operator	Description
(?:)	Conditional Operator returns Value based on the condition. It is like if - else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property.
instanceof	Checks if the object is an instance of given type
new	Creates an instance
typeof	Checks the type of object
void	it discards the expression's return value.
yield	Checks what is returned in a generator by the generator's interator.

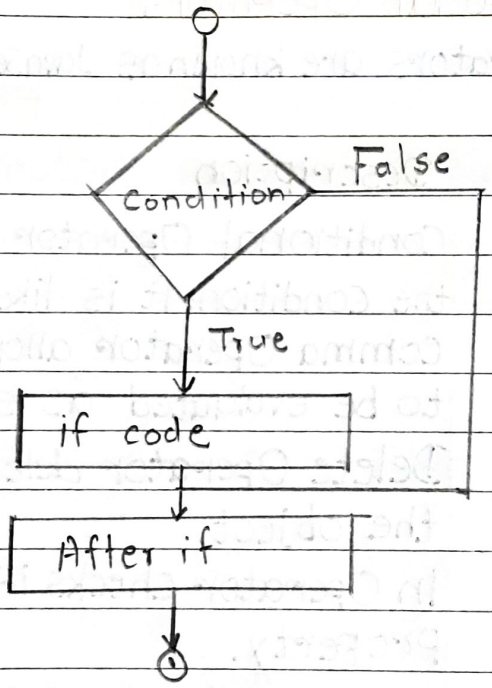
JavaScript If - else

The JavaScript if-else statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

1. If statement
2. If else statement
3. If else if statement

JavaScript If Statement

It evaluates the Content only if expression is true.

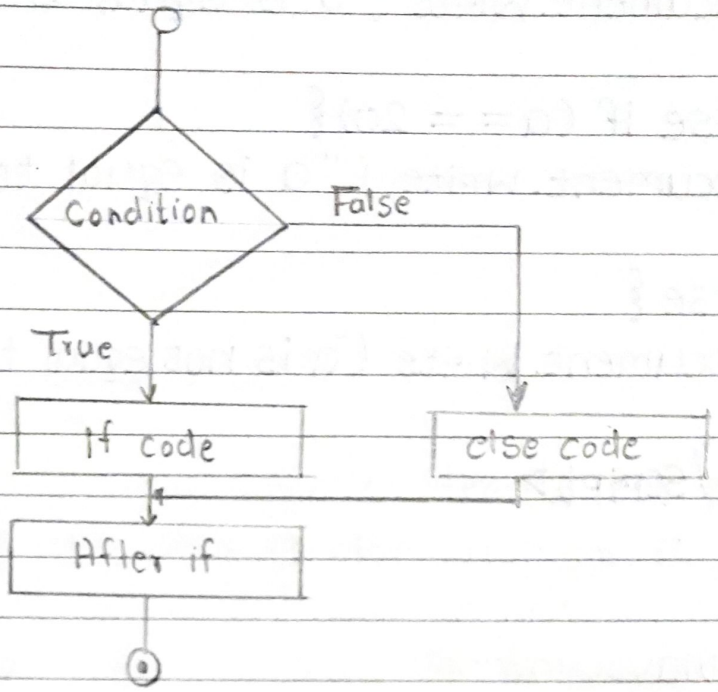


Example

```
<script>
var a = 20;
if (a>10){
document. Write ("Value of a is grater than 10");
}
</script>
```

JavaScript If... else statement

- If evaluates the content whether condition is true of false.



Example

```
<Script>
Var a = 20;
if (a % 2 = 0) {
document.Write ("a is even number");
}
else {
document. Write ("a is odd number");
}
</Script>
```

JavaScript If... else if statement

It evaluates the content only if expression is true.

Example

```
<script>
Var a = 20;
if (a= = 10) {
document. Write ("a is equal to 10");
```

```
}
else if (a = = 15) {
document.Write ("a is equal to 15");
}
else if (a= = 20){
document. Write ("a is equal to 20");
}
else {
document. Write ("a is not equal to 10, 15 or 20");
}
</script>
```

Javascript Switch

The Javascript switch statement is used to execute one code from multiple expressions.

Example

```
<script>
Var grade = 'B';
Var result;
Switch(grade){
Case 'A';
result = "A Grade";
break;
Case 'B';
result = "B Grade";
break
Case 'C';
result = "C Grade";
break;
default;
```

```
result = " No Grade";
}
document. Write (result);
</script>
```

Output
B Grade

JavaScript Loops

The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
There are four types of loops in JavaScript.

1) JavaScript For loop

The JavaScript for loop iterates the elements for the fixed number of times. It should be used if number of inte-ration is known.

Example

```
<script>
for (i= 1;i<=5; i++)
{
document.Write(i+ "<br />")
}
</script>
```

Output
1
2
3
4
5

2) JavaScript While loop

The JavaScript while loop iterates the elements for the infinite number of times. It should be used if number of iteration is not known.

Example

```
<script>
Var i = 11;
While (i<= 15)
{
document. Write (i+"<br/>");
i++;
}
</script>
```

Output

```
11
12
13
14
15
```

3) JavaScript do while loop

The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false.

Example

```
<script>
Var i = 21;
do {
document. Write (i+ "<br/> ");
i++;
} While (i<=25);
</Script>
```

Output

```
21
22
23
24
25
```

JavaScript Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

1. Code reusability
2. Less coding

JavaScript Function Example

```
< script >
function msg (){
alert ("hello! this is message");
}
</ script >
<input type = "button" onclick ="msg()" value= "Call function"/>
```

Output call function

JavaScript Function Arguments

We can call function by passing arguments.

Example

```
<script >
function getcube(number){
alert(number* number * number);
}
</script >
<form >
<input type = "button" = "click" onclick = "getcube(4)"/>
</form>
```

Output Click

JavaScript Function Object

In JavaScript, the purpose of Function constructor is

to create a new Function object. It executes the code globally. However, if we call the constructor directly, a function is created dynamically but in an unsecured way.

JavaScript Function Methods

Method	Description
apply()	It is used to call a function contains this value and a single array of arguments.
bind()	It is used to create a new function.
Call()	It is used to call a function contains this value and an argument list.
to String()	It returns the result in a form of a string.

Example

```
<Script>
Var add =new Function("num1","num2","return num1+num2");
document.Write ln (add (2,5));
</ script>
```

Output 7

JavaScript Objects

JavaScript is an object-based language. Everything is an object in JavaScript.
JavaScript is template based not class based. Here, we don't create class to get the object. But, we direct create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1) JavaScript Object by object literal

The syntax of creating object using object literal.

```
Object = {property 1; Value 1, property 2; value 2 .... propertyN: ValueN}
```

2) By creating instance of Object

The syntax of creating object directly.

```
Var objectname = new Object();
```

Here, new keyword is used to create object.

3) By using an object constructor

Here, you need to create function with arguments. Each argument value can be assigned in the current object by using this keyword.
The this keyword refers to the current object.

Example

```
<script>
function emp (id,name, salary){
this. id = id;
this. name = name;
this. Salary = salary;
}
e = new emp(103, "Vimal Jaiswal", 30000);
document. write (e.id + " " + e.name + " " + e. salary);
</script>
```

Output 103 Vimal Jaiswal 30000

JavaScript Object Methods

1. Object.assign()
This method is used to copy enumerable and own properties from a source object to a target object.

2. Object.create()
This method is used to create a new object with specified prototype object and properties.

3. Object.defineProperty()
This method is used to describe some behavioral attributes of the property.

4. Object.defineProperties()
This method is used to create or configure multiple object properties.

5. Object.entries()
This method returns an array with arrays of the key, value pairs.

6. Object.freeze()
This method prevents existing properties from being removed.

7. Object.getOwnPropertyDescriptor()
This method returns a property descriptor for the specified property of the specified object.

8. Object.getOwnPropertyDescriptors()
This method returns all own property descriptors of a

given object.

9. Object.getOwnPropertyNames()
The method returns an array of all properties found.

10. Object.getOwnPropertySymbols()
This method returns an array of all own symbol key properties.

11. Object.getPrototypeOf()
This method returns the prototype of the specified object.

12. Object.is()
This method determines whether two values are the same value.

13. Object.isExtensible()
This method determines if an object is extensible.

14. Object.isFrozen()
This method determines if an object was frozen.

15. Object.isSealed()
This method determines if an object is sealed.

16. Object.keys()
This method returns an array of a given object's own property names.

17. Object.preventExtensions()
This method is used to prevent any extensions of an object.

18. Object.setPrototypeOf()
This method sets the prototype of a specified object to another object.

19. Object.seal()
This method prevents new properties from being added and marks all existing properties as non-configurable.

20. Object.values()
This method returns an array of values.

JavaScript Array

JavaScript array is an object that represents a collection of similar type of elements.
There are 3 ways to construct array in JavaScript

1) JavaScript array literal

```
var arrayname = [value1, value2.....valueN];
```

2) JavaScript Array directly

```
var arrayname = new Array();
```

Here, new keyword is used to create instance of array.

3) JavaScript array constructor

Here, you need to create instance of array by passing arguments in constructor so that we don't have to provide value explicitly.

Example
<script>

```
Var emp = new Array("Jai", "Vijay", "Smith");
for (i=0; i< emp. length; i++) {
document. Write ( emp [i] + "<br>");
}
</ script>
```

Output
Jai
Vijay
Smith

JavaScript Array Methods

1. concat ()
It returns a new array object that contains two or more merged arrays.

2. copyWithin()
It copies the part of the given array with its own elements and returns the modified array.

3. entries()
It creates an iterator object and a loop that iterates over each key/value pair.

4. every()
It determines whether all the elements of an array are staisfying the provided function conditions.

5. flat()
It creates a new array carrying sub-array elements concatenated recursively till the specified depth.

6. flatMAP()
It maps all array elements via mapping function, than flattens the result into a new array.

7. fill()
It fills elements into an array With static values.

8. forEach()
It invokes the provided function once for each element of an array.

9. includes()
It checks Whether the given array Contains the Specified element.

10. isArray()
It tests if the passed value ia an array.

11. join()
It joins the elements of an array as a String.

12. push()
It adds one or more elements to the end of an array.

JavaScript string

The JavaScript string is an object that represents a sequence of Characters.
There are 2 Ways to create String in JavaScript

1) By string literal
The String literal is created using double quotes.

Var String name = "string value";

2) By string object

The syntax of creating string object using new keyword.

Var stringname = new String ("string literal");

JavaScript String Methods

1. char At ()
It provides the char value present at the specified index.

2. Char Code At()
It provides the Unicode value of a character present at the specified index.

3. concat()
It provides a combination of two or more strings.

4. index Of()
It provides the position of a char value present in the given string.

5. lastIndex Of()
It provides the position of a char value present in the given string by searching a character from the last position.

6. Search()
It searches a specified regular expression in a given string and returns its position if a match occurs.

7. match()

It searches a specified regular expression in a given string and returns that regular expression if a match occurs.

8. replace()
It replaces a given string with the specified replacement.

9. substr()
It is used to fetch the part of the given string on the basis of the specified starting position and length.

10. substring()
It is used to fetch the part of the given string on the basis of the specified index.

11. toLowerCase()
It converts the given string into lowercase letter.

12. toLocaleLowerCase()
It converts the given string into lowercase letter on the basis of host?s current locale.

13. toUpperCase()
It converts the given string into uppercase letter.

JavaScript Date Object

The JavaScript date object can be used to get year, month and day. You can display a timer on the webpage by the help of JavaScript date object.

Constructor

You can use 4 Variant of Date constructor to create date object.

1. Date()
2. Date (milliseconds)
3. Date (date String)
4. Date (year, month, day, hours, minutes, seconds, milliseconds).

JavaScript Date Methods

1. get Date()
It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of local time.

2. get Day()
It returns the integer value between 0 and 6 that represents the day of the week on the basis of local time.

3. get Full Years()
It returns the integer value that represents the year on the basis of local time.

4. get Hours()
It returns the integer value between 0 and 23 that represents the hours on the basis of local time.

5. get Milliseconds()
It returns the integer value between 0 and 999 that represents the milliseconds on the basis of local time.

6. get Minutes()
It returns the integer value between 0 and 59 that represents the minutes on the basis of local time.

7. get Month()
It returns the integer value between 0 and 11 that represents the month on the basis of local time.

8. get Seconds()
It returns the integer value between 0 and 60 that represents the seconds on the basis of local time.

9. get UTC Date()
It returns the integer value between 1 and 31 that represents the day for the specified date on the basis of universal time.

10. get UTC Day()
It returns the integer value between 0 and 6 that represents the day of the week on the basis of universal time.

11. get UTC Full Years()
It returns the integer value that represents the year on the basis of universal time.

JavaScript Math Methods

The JavaScript math object provides several constants and methods to perform mathematical operation. Unlike date object, it doesn't have constructors.

1. abs()
It returns the absolute value of the given number.

2. acos()
It returns the arccosine of the given number in radians.

3. asin()
It returns the arcsine of the given number in radians.

4. atan()
It returns the arc-tangent of the given number in radians.

5. cbrt()
It returns the cube root of the given number.

6. ceil()
It returns a smallest integer value, greater than or equal to the given number.

7. cos()
It returns the cosine of the given number.

8. cosh()
It returns the hyperbolic cosine of the given number.

9. exp()
It returns the exponential form of the given number.

10. floor()
It returns largest integer value, lower than or equal to the given number.

11. hypot()
It returns square root of sum of the squares of given numbers.

12. log()
It returns natural logarithm of a number.

13. max()
It returns maximum value of the given numbers.

14. min()
It returns minimum value of the given numbers.

15. pow()
It returns value of base to the power of exponent.

16. random()
It returns random number between 0 (inclusive) and 1 (exclusive).

17. round()
It returns closest integer value of the given number.

18. sign()
It returns the sign of the given number.

19. sin()
It returns the sine of the given number.

20. sinh()
It returns the hyperbolic sine of the given number.

21. sqrt()
It returns the square root of the given number.

22. trunc()
It returns an integer part of the given number.

JavaScript Number Object

The JavaScript number object enables you to represent a numeric value. It may be integer to floating-point. JavaScript number object follows IEEE standard to represent the floating-point numbers.

```
Varn = new Number (value);
```

JavaScript Number Constant

Constant	Description
MIN_VALUE	returns the largest minimum value.
MAX_VALUE	returns the largest maximum value.
POSITIVE_INFINITY	returns positive infinity, overflow value.
NEGATIVE_INFINITY	returns negative infinity, overflow value.
NaN	represents "Not a Number" value.

JavaScript Number Methods

1. isFinite()
It determines whether the given value is a finite number.

2. isInteger()
It determines whether the given value is an integer.

3. parseFloat()
It converts the given string into a floating point number.

4. parseInt()
It converts the given string into an integer number.

5. toExponential()
It returns the string that represents exponential notation of the given number.

6. toFixed()
It returns the string that represents a number with exact digits after a decimal point.

7. toPrecision()
It returns the string that represents a number of specified precision.

8. toString()
It returns the given number in the form of string

JavaScript Boolean

JavaScript Boolean is an object that represents value in two states; true or false. You can

Create the JavaScript Boolean object by Boolean() constructor.

```
Boolean b = new Boolean(value);
```

Example

```
<script>
document.Write (10<20); // true
document.Write (10< 5); // false
</script>
```

JavaScript Boolean Properties

Property	Description
constructor	returns the reference of Boolean function that created Boolean object.
prototype	enables you to add properties and methods in Boolean prototype.

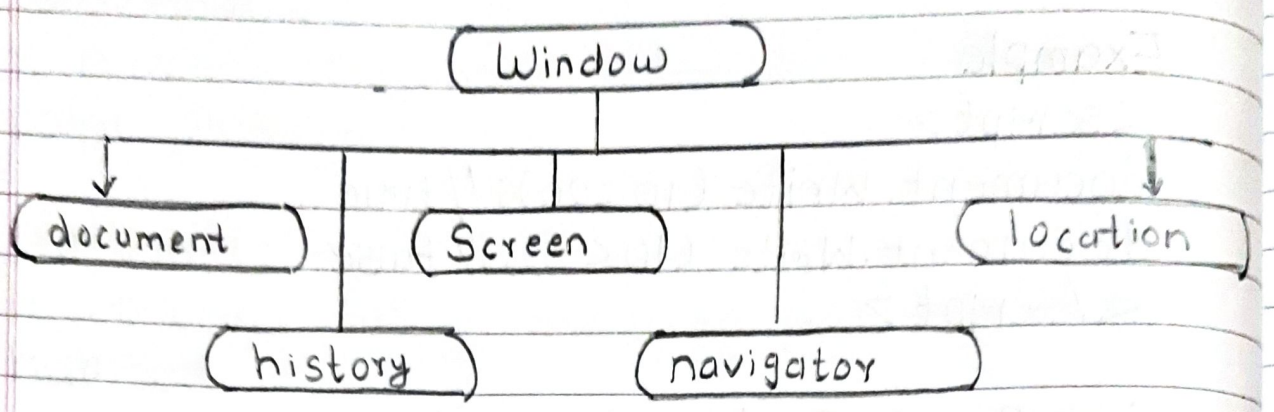
JavaScript Boolean Methods

Method	Description
to Source()	returns the source of Boolean object as a string.
to String()	Converts Boolean into string.
Value Of()	Converts other type into Boolean.

Browser Object Model

The Browser Object Model (BOM) is used to interact with the browser.

Window.alert ("hello javatpoint");
is same as:
alert ("hello javatpoint");



Window Object

The window object represents a window in browser. An object of window is created automatically by the browser.

Example of alert() in javascript

```
<Script type = "text / java script ">
function msg(){
alert ("Hello Alert Box");
}
</Script>
<input type = "button" Value ="Click"onclick="msg()"/>
```

Output click

Example of Confirm() in javascript

```
<script type = "text/javascript">
function msg (){
Var V = Confirm ("Are u sure?");
```

```
if (v==true){
alert("ok");
}
else {
alert("cancel");
}
}
</script>
<input type="button" value="delete record" onclick="msg()"/>
```

Output delete record

Example of prompt() in javascript

```
<script type = "text/javascript">
function msg(){
Var v = prompt("Who are you?");
alert("I am " + v);
}
</script>
<input type="button" value="click" onclick = "msg()"/>
```

Output click

Example of open() in javascript

```
<script type = "text/javascript">
function msg(){
open("http://www.javatpoint.com");
}
</script>
<input type="button" value = "javat point" onclick = "msg()"/>
```

Output javat point

Example of setTimeout() in javascript

```
<script type = "text/javascript">
function msg() {
set Time out (
function() {
alert ("Welcome to Javatpoint after 2 seconds")
}, 2000);
}
</script>
<input type = "button" value = "click" onclick = "msg()"/>
```

Output click

JavaScript History Object

The JavaScript history object represents an array of URLs visited by the user. By using this object, you can load previous, forward or any particular page.

Window. history
Or,
history

Property of JavaScript Object

NO.	Property	Description
1.	length	returns the length of the history URLs.

Methods of JavaScript history object

1. forward()
loads the next page.

2. back()
loads the previous page.

3. go()
loads the given page number.

JavaScript Navigator Object

The JavaScript navigator object is used for browser detection. It can be used to get browser information such as appName, appCodeName, userAgent etc.

Window.navigator
Or
navigator

Property of JavaScript navigator object

No.	Property	Description
1.	appName	returns the name
2.	appVersion	returns the version
3.	appCodeName	returns the code name
4.	cookieEnabled	returns true if cookie is enabled otherwise false
5.	userAgent	returns the user agent
6.	language	returns the language. It is supported in Netscape and Firefox only.
7.	userLanguage	returns the user language. It is supported in Netscape and Firefox only.
8.	plugins	returns the plugins. It is supported in Netscape and Firefox only.
9.	systemLanguage	returns the system language. It is supported in IE only.

10.	mimeTypes[]	returns the array of mime type. It is supported in Netscape and Firefox only.
11.	platform	returns the platform e.g. Win32.
12.	Online	returns true if browser is online otherwise false.

Methods of JavaScript navigator object

NO.	Method	Description
1.	javaEnabled()	Checks if java is enabled.
2.	taintEnabled()	Checks if taint is enabled. It is deprecated since JavaScript 1.2.

JavaScript Screen Object

The JavaScript screen object holds information of browser screen. It can be used to display screen width, height, colorDepth, pixelDepth etc.

```
window.screen
```

Or,

```
screen
```

Property of JavaScript Screen Object

NO.	Property	Description
1.	width	returns the width of the screen.
2.	height	returns the height of the screen.
3.	availWidth	returns the available height.
4.	availHeight	returns the available height.

5.	colorDepth	returns the color depth.
6.	pixelDepth	returns the pixel depth.

Example

```
<script>
document. Writeln("<br/>screen.Width:"+screen.Width);
document. Writeln ("<br/>screen.height:"+screen.height);
document. Writeln ("<br/>screen.availWidth:"+screen.availWidth);
document.Writeln ("<br/>Screen.availHeight:"+screen. vav
document.Writeln("<br/>screen color Depth:"+screen color Depth);
document. Writeln ("<br/> screen pixel Depth: "+screen pixel Depth:)
</script>
```

Document Object Model

The document object represents the whole html document. When html document is loaded in the browser, It becomes a document object. It is the root element that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

Window. document
is same as
document

Methods of document object

We can access and change the contents of document by its methods.

Method	Description
• Write ("String")	Writes the given string on the document.
• Writeln("string")	Writes the given string on the document with newline character at the end.
• getElementById()	returns the element having the given id value.
• getElements ByName()	returns all the elements having the given name value.
• getElements By Tag Name()	returns all the elements having the given tag name.
• getElements By Class Name()	returns all the elements having the given class name.

Accessing field value by document object

Example

```
<script type = "text/javascript">
function printvalue(){
Varname = document. form 1.name.value;
alert("Welcome;" + name);
}
</script>
<form name = "form 1">
Enter Name; <input type = "text" name = "name"/>
<input type = "button" onclick= "printvalue()" value="Printname"/>
</form>
```

Output

Enter Name: [ ] Print name

JavaScript - document.getElementById() method

The document.getElementById() method returns the element of specified id.

In the previous page, we have used document.form1.name.value to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Example

```
<script type = "text/javascript">
function getcube(){
var number = document.getElementById("number").value;
alert(number * number * number);
}
</script>
<form>
Enter No: <input type = "text" id = "number" name = "number"/><br/>
<input type = "button" value = "cube" onclick = "getcube()"/>
</form>
```

Output :

Enter No: [ ]

Cube

Get Elements By Class Name ()

The getElementsByClassName() method is used for selecting or getting the elements through their class name value. This DOM method returns an array-like object that consists of all the elements having the specified classname. On calling the getElementByClassName() method or any particular element, it will search the whole document and will

return only thouse elements which match the specified or given class name.

```
Varele = document.getElementsByClassName('name');
```

Example

```
<html>
<head><h5> DOM Methods </h5></head>
<body>
<div class = "class">
This is a simple class implementation
</div>
<Script type = "text/javascript">
Varx = document.getElement ByClassName('class');
document.Write("On Calling x, It will return an array-
      like object: <br>"+x);
</script>
</body>
</html>
```

Output:

```
DOM Methods
This is a simple class implementation
On callingx, It will return an array-like object:
      [Object HTML Collection]
```

JavaScript - document.getElementsByName() method

The document.getElementsByName() method returns all the element of specified name.

```
document.getElements By Name("name")
```

Example

```
<script type = "text/javascript">
function totalelements()
{
Var allgenders = document.getElementsByName ("gender");
alert ("Total Genders."+ allgenders.length);
}
</script>
<form>
make:<input type = "radio" name = "gender"value="male" >
Female:<input type = "radio" name = "gender"value = "female">
<input type = "button"onclick "totalelements()"value = "Total Genders">
</form>
```

Output

Male: 0 Female: 0 Total Genders

JavaScript - document.get Elements By Tag Name() method

The document.getElements By Tag Name () method returns all the element of specified tag name.

```
document.getElements By TagName("name")
```

Example

```
<script type = "text/javascript">
function countpara(){
Var totalpara = document.getElementsByTagName("p");
alert("total ptags are:"+ totalpara.length);
}
</script>
<P> This is a paragraph</p>
<P> Here we are going to count total number of
  Paragraphs by getElements By Tag Name()method.</p>
<P> Let's see the simple example</p>
<button onclick = "countpara()">Count paragraph</button>
```

Output

This is a pragraph.
Here we are going to count total number of paragraphs by getElementsByTagName()method.
Let's see the simple example.

Count Paragraph

JavaScript - innerHTML

The innerHTML property can be used to write the dynamic html on the html document.
It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

Example

```
<script type = "text/javascript">
```

```
function Showcommentform(){
Var data = "Name; <input type = 'text' name = 'name'><br>Comment:
<br>textarea rows = '5' cols = '80'></textarea>
<br><input type = 'submit' Value = 'Post Comment'>";
documet.getElementby Id ('mylocation'). innerHTML = data;
}
</Script>
<form name = "my Form">
<input type = "button" value = "comment" onclick = "Showcommentform()">
<div id = "mylocation"></div>
</form>
```

Output

comment

JavaScript Form Validation

It is important to Validate the form submitted by the user because it Can have inappropriate values. So, Validation is must to authenticate user.

JavaScript provides facility to validate the form on the Client - Side. So data processing Will be faster than Server - Side Validation. Most of the Web developers prefer JavaScript form Validation.

Through Javascript, We Can Validate name, password, email, date, mobile numbers and more fields.

Example

```
<Script>
function validateform(){
Var name = document.my form.name.Value;
```

```
Var password = document.myform.password.value;
if (name == null || name == "") {
 alert ("Name can't be blank");
 return false;
} else if (password.length<6) {
 alert ("Password must be at least 6 characters long.");
 return false;
 }
}
</script>
<body>
<form name = "myform" method = "post" action = "abc.jsp"
   onsubmit = "return validate form()">
Name: <input type = "text" name = "name" = "password"><br/>
Password: <input type = "password" name = "password"><br/>
 <input type = "submit" value = "register">
  </form>
```

JavaScript email Validation

We can validate the email by the help of Java Script.

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after (dot).

Example

```
<script>
function validate email()
```

```
{
Var x = document.myform.email.Value;
Var atposition = x.index Of("@");
Var dot position = x.lastIndexOf(".");
if(atposition<1 || dotposition<atposition+2||dotposition+2==x.length){
alert("Please enter a Valid email address \n atpostion;"+
    atposition+"\n dotposition."+ dotposition);
return false;
  }
}
</script>
<body>
<form name ="my form" method ="Post" action = "#"
        onsubmit = "return Validateemail();">
Email: <input type = "text" name = "email"><br/>
      <input type = "submit" Value = "Register">
      </form>
```

JavaScript Classes

In JavaScript, classes are the special type of functions. We can define the class just like function declearations and function expressions.

The JavaScript class contains various class members within a body including methods or constructor. The class is executed in strict mode. So, the code containing the silent error or mistake throws an error.

The class syntax contains two components:

- Class declearation
- Class expressions

Class Declarations

A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.

Example

```
<script>
//Declaring class
 Class Employee
 {
//Initializing an object
  constructor(id, name)
  {
  this.id = id;
 this.name = name;
 }
//Declaring method
  detail()
  {
document.writeln(this.id+" "+ this.name + "<br>")
   }
  }
//passing object to a variable
Var e1 = new Employee(101, "Martin Roy");
Var e2 = new Employee(102, "Duke William");
e1.detail(); //calling method
e2.detail();
</script>
```

Output: 101 Martin Roy
102 Duke William

Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class. So, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However this will not be possible with class declaration.

Unnamed Class Expression

The class can be expressed without assigning any name to it.

example

```
<script>
Var emp = class {
 constructor(id, name) {
 this. id = id;
 this. name = name;
 }
};
document. Write in (emp. name)
 </script>
```

Output:

emp

JavaScript Objects

A javascript object is an entity having state and behavior. JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't

Class expressions

Another way to define a class is by using a class expression. Here, it is not mandatory to assign the name of the class so, the class expression can be named or unnamed. The class expression allows us to fetch the class name. However this will not be possible with class declaration.

Unnamed Class Expression

The class can be expressed without assigning any name to it.

example

```
<script>
Var emp = class {
 constructor(id, name){
 this.id = id;
 this.name = name;
 }
};
document.Writeln(emp.name)
 </script>
```

Output:

emp

JavaScript Objects

A javascript object is an entity having state and behavior. JavaScript is an object-based language. Everything is an object in JavaScript.

JavaScript is template based not class based. Here, we don't

Create class to get the object. But we direct Create objects.

Creating Objects in JavaScript

There are 3 ways to create objects.

1. By object literal
2. By Creating instance of Object directly
3. By using an object constructor

1) JavaScript Object by object literal

Object = {property1.Value1.property2.Value2.... propertyN.Value N}

Example

```
<script>
emp = {id:102,name."Shyam Kumar", Salary:40000}
document.Write (emp.id+" " +emp.name+" "+emp.salary);
</script>
```

Output

102 Shyam Kumar 40000

2) By creating instance of Object

Var objectname = new Object();

Example

```
<script>
Var emp = new object();
 emp.id = 101;
```

```
emp.name = "Ravi Malik"
emp.salary = 50,000;
document.Write(emp.id + " " + emp.name + " " + emp.Salary);
</script>
```

Output
101 Ravi 50000

3.) By using an Object constructor:
Here, you need to create function with arguments. Each argument value can be assigned in the current object by using the keyword.

example

```
<script>
function emp(id, name, salary){
this.id = id;
this.name = name;
this.salary = salary;
}
e=new emp(103, "Vimal Jaiswal", 30000);
document.write(e.id + " " + e.name + " " + e.salary);
</script>
```

Output
103, Vimal Jaiswal 30000

JavaScript Object Methods

1. Object.assign()
This method is used to copy enumerable and own properties from a source object to a target object.

2. Object.create()
This method is used to create a new object with the specified prototype object and properties.

3. Object.defineProperty()
This method is used to describe some behavioral attributes of the property.

4. Object.defineProperties()
This method is used to create or configure multiple object properties.

5. Object.entries()
This method returns an array with arrays of the key. value pairs.

6. Object.freeze()
This method prevents exisiting properties from being removed.

7. Object.getOwnProperty Descriptor()
This method returns a property descriptor for the specified property of the specified object.

8. Object.get Own Property Descriptors()
This method returns all own property descriptors of a given objects.

9. Object.get Own Property Named()
This method returns an array of all properties found.

10. Object.getOwnPropertySymbols()
This method returns an array of all own symbol key properties.

11. Object.getPrototypeOf()
This method returns the prototype of the specified object.

12. Object.is()
This method determines whether two values are the same value.

13. Object.isExtensible()
This method determines if an object is extensible.

14. Object.isFrozen()
This method determines if an object was frozen.

15. Object.isSealed()
This method determines if an object is sealed.

16. Object.keys()
This method returns an array of a given objects own property names.

17. Object.preventExtensions()
This method is used to prevent any extensions of an object.

18. Object.seal()
This method prevents new properties from being added and marks all existing properties as non-configurable.

19. Object.setPrototypeOf()
This methods sets the prototype of a specified object to another object.

20. Object.values()
This method returns an array of values.

JavaScript Prototype Object

Javascript is a prototype-based language that facilitates the objects to acquire properties and features from one another. Here, each object contains a prototype object.

ClassName.prototype.methodName

Prototype chaining

In JavaScript, each object contains a prototype object that acquires properties and methods from it. Again an objects prototype object may contain a prototype object that also acquires properties and methods, and so on. It can be seen as prototype chaining.

Example

```
<Script>
function Employee (firstName, lastName)
  {
  this.firstName = firstName;
  this.lastName = lastName;
}
Employee.prototype.fullName = function()
```

```
{
return this. first Name +" " + this last Name;
}
Var employee1 = new Employee ("Martin", "Roy")
Var employee2 = new Employee ("Duke", "William")
document. Writeln (employee1. fullName() + "<br>");
document. Writeln (employee2. full Name());
</script>
```

Output

Martin Roy

Duke William

JavaScript Constructor Method

A JavaScript constructor method is a special type of method which is used to initialize and create an object. It is called when memory is allocated for an object.

Points to remember

- The constructor keyword is used to declare a constructor method.
- The class can contain one constructor method only.
- JavaScript allows us to use parent class constructor through super keyword.

Constructor Method Example

```
<script>
Class Employee {
 Constructor() {
 this. id =101;
this. name = " Martin Roy";
 }
}

Var emp = new Employee();
document. Writeln (emp.id +" " +emp.name);
 </script>
```

Output

101 Martin Roy

Constructor Method Example: super Keyword

```
<Script>
Class CompanyName
 {
 Constructor()
 {
this. company ="Javatpoint";
 }
 }
 Class Employee extends Company Name {
  Constructor (id, name) {
  Super();
  this.id = id;
  this. name = name;
  }
}
Var emp = new Employee (1,"John");
```

```
document.writeln(empid+" "+emp.name+" "+emp.company);
</script>
```

Output
1 John Javatpoint

JavaScript static Method

The JavaScript provides static methods that belong to the class instead of an instance of that class. So, an instance is not required to all the static method. These methods are called directly on the class itself.

Points to remember

- The static keyword is used to declare a static method.
- The static method can be of any name.
- A class can contain more than one static method.
- If we declare more than one static method with a similar name, the JavaScript always invokes the last one.
- The static method can be used to create utility functions.
- We can use this keyword to call a static method within another static method.
- We cannot use this keyword directly to call a static method within the non-static method. In such case, we can call the static method either using the class name or as the property of the constructor.

Example

```
<script>
class Test
```

```
{
Static display()
{
return "Static method is invoked"
}
}
document. Writeln (Test display());
</Script>
```

Output
Static method is invoked

Javascript Encapsulation

The Javascript Encapsulation is a process of binding the data with the functions acting on the data. It allows us to control the data and validate.
To achieve an encapsulation in JavaScript:-

- Use var keyword to make data members private.
- Use setter methods to set the data and getter methods to get the data.

The encapsulation allows us to handle an object using the following properties:

Read / Write -
Here, we use setter methods to write the data and getter methods read that data.

Read Only -
In this case, we use getter methods only.

Write Only -
In this case, we use setter methods only.

Example

```
<script>
Class Student
{
Constructor()
{
Var name;
Var marks;
}
get name()
{
return this.name;
}
SetName(name)
{
this.name = name;
}
get Marks ()
{
return this. marks
}
setMarks(Marks)
{
this. marks = marks
}
}
Var stud = new Student.();
Stud. setName ("John");
Stud.setMarks (80);
document. Writeln(stud.getName()+" "+Studgetmarks());
</script>
```

Output: John 80

JavaScript Inheritance

The JavaScript inheritance is a mechanism that allows us to create new classes on the basis of already existing classes. It provides flexibility to the child class to reuse the methods and variables of a parent class.
The Javascript extends keyword is used to create a child class on the basis of a parent class. It facilitates child class to acquire all the properties and behavior of its parents class.

Points to remember

- It maintains an IS-A relationship.
- The extends keyword is used in class expressions or class declarations.
- Using extends keyword, we can acquire all the properties and behavior of the inbuilt object as well as custom classes.
- We can also use a prototype-based approach to achieve inheritance.

JavaScript extends Example: inbuilt object

```
<script>
class Moment extends Date {
 constructor(){
   super();
 }}
 var m = new Moment();
document.writeln("Current date")
document.writeln(m.getDate()+"-"+(m.getMonth()+1)
  + "-" + m.getFullYear());
```

```
</script>
```

Output:
Current date: 31 - 8 - 2018

JavaScript Polymorphism

The polymorphism is a core concept of an object-Oriented paradigm that provides a way to perform a single action in different forms. It provides an ability to call the same method on different JavaScript objects. As JavaScript is not a type-safe language, we can pass any type of data members with the methods.

Example

```
<Script>
 Class A
  {
   display()
   {
   document.Writeln("A is invoked");
   }
  }
   Class B extends A
   {
   }
  Var B = new B()
   b. display();
  </script>
```

Output:
A is invoked

JavaScript Abstraction

An abstraction is a way of hiding the implementation details and showing only the functionality to the users. In other words, it ignores the irrelevant details and shows only the required one.

Points to remember

- We cannot create an instance of Abstract Class.
- It reduse the duplication of code.

Example

```
<script>
//Creating a constructor function
 function Vehicle()
 {
  this.vehicleName = "Vehicle Name";
  throw new Error("You cannot Create an instance of
      Abstract class");
 }
  Vehicle.prototype.display = function()
  {
  return "Vehicle is : " + this.vehicleName;
 }
//creating a constructor function
 function Bike(vehicleName)
 {
  this.vehicleName = vehicleName;
 }
//Creating object without using the function constructor
 Bike.prototype = Object.create(Vehicle.prototype);
 var bike = new Bike ("Honda");
```

```
document.writeln(bike.display());
</script>
```

Output:
Vehicle is: Honda

Java Script Cookies

A cookie is an amount of information that persists between a server-side and a client-side. A web browser stores this information at the time of browsing.
A cookie contains the information as a string generally in the form of a name-value pair separated by semi-colons. It maintains the state of a user and remembers the users information among all the web pages.

How Cookies Works?

- When a user sends a request to the server, then each of that request is treated as a new request sent by the different user.
- So, to recognize the old user, we need to add the cookie with the response from the server.
- browser at the client-side.
- Now, whenever a user sends a request to the server, the cookie is added with that request automatically. Due to the cookie, the server recognizes the users.

How to create a Cookie in JavaScript?

In Javascript, we can create, read, update, and delete a cookie by using document.cookie property.

```
document.cookie = "name = value";
```

Example

```
<!DOC TYPE html>
<html>
<head>
</head>
<body>
<input type="button" value = "setCookie" onclick=
    " setCookie()">
<input type="button" value= "get Cookie" onclick=
    " getCookie()">
<script>
function setCookie()
{
document. cookie = "username = Duke Martin";
}
function getCookie()
{
if(document.cookie.length!=0)
{
alert (document. cookie);
}
else {
alert ("Cookie not available");
}
}
```

```
</script>
</body>
</html>
```

Cookie Attributes

JavaScript provides some optional attributes that enhance the functionality of Cookies. Here, is the list of some attributes with their description.

Attributes	Description
expires	It maintains the state of a cookie up to the specified date and time.
max-age	It maintains the state of a cookie upto the specified time. Here, time is given in seconds.
Path	It expands the scope of the cookie to all pages of a website.
domain	It is used to specify the domain for which the cookie is vaild.

Cookie expires attribute

The cookie expires attribute provides one of the ways to create a persistent cookie. Here, a date and time are declared that represents the active period of a cookie. Once the declared time is passed, a cookie is deleted automatically.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
```

```
<body>
<Input type = "button" value = "set Cookie" onclick =
        " set cookie()">
<input type = "button" value = "get cookie" onclick =
        " get Cookie()">
<Script>
function setCookie()
{
document.cookie = "username = Duke Martin; expires =
    Sun, 20 Aug 2030 12:00:00 UTC";
}
function getCookie()
{
  if(document.cookie.length!=0)
  {
   Var array = document.cookie.split("=");
 alert ("Name = " + array[0] + " " + "Value=" + array[1]);
  }
  else
  {
   alert ("Cookie not available");
   }
}
  </script>
 </body>
 <html>
```

Cookie with multiple Name-Value pairs

In JavaScript, a cookie can contain only a single name-Value pair. However, to store more than one name-Value pair, we can use the following approach:-

- Serialize the custom object in a JSON string, parse it then store in a cookie.
- For each name-value pair, use a separate cookie.

Examples to store Name-Value Pair in a cookie

Example

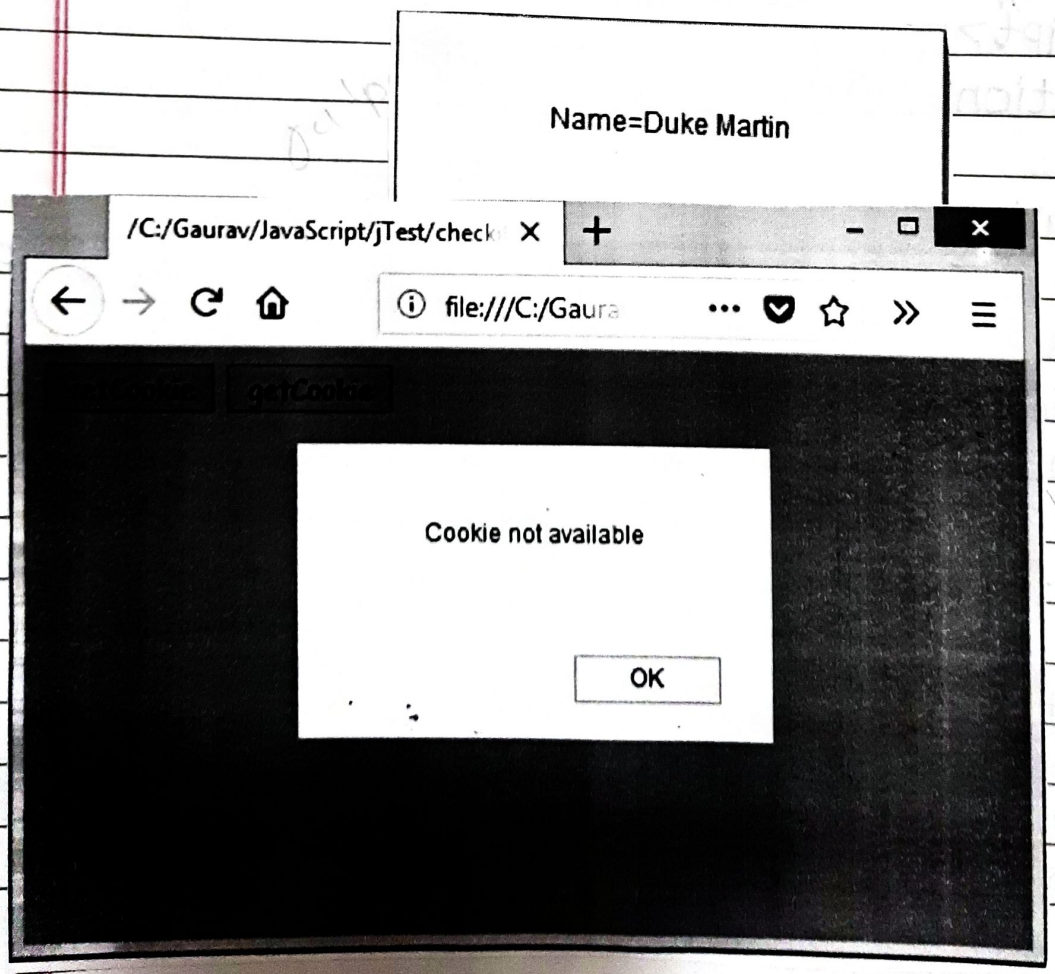
```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
  Name : <input type = "text" id = "name"><br>
  Email : <input type = "email" id = "email"><br>
  Course : <input type = "text" id = "course"><br>
<input type = "button" value = "Set Cookie" onclick = "SetCookie()">
<input type = "button" value = "Get Cookie Onclick = "getCookie()">
<Script>
 function set Cookie ()
 {
//Declaring 3-key value pairs
  Var info = "Name = " + document.getElement By Id("name").
  Value + "Email = " + document.get Element B
//Providing all 3 key-value pairs to a Single cookie
  document. cookie = info;
 }
 function get cookie ()
 {
  if(document. cookie.length! = 0)
  {
 //invoking key-value pair stored in a cookie
  alert (document. cookie);
```

```
}
else
{
alert ("cookie not available")
}
}
</script>
<body>
<html>
```

Output

Name: Duke Martin
Email: duke@gmail.com
Course: JavaScript
Set Cookie | Get Cookie

On clicking Get Cookie button, the below dialog box appears.



Deleting a Cookie in JavaScript

Different Ways to delete a Cookie

There are the following ways to delete a cookie:

- A cookie can be deleted by using expire attribute.
- A cookie can also be deleted by using max-age attribute.
- We can delete a cookie explicitly, by using a web browser.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<input type="button" value = "Set Cookie" onclick="SetCookie()">
<input type="button" value = "Get Cookie" onclick=getCookie()>
<script>
function setCookie()
{
  document.cookie = "name =Martin Roy; expires = Sun,
        20 Aug 2000  12:00:00  UTC";
}
function getCookie()
{
    if(document.cookie.length!=0)
    {
    alert(document.cookie);
    }
    else
    {
```

```
        alert ("cookie not avaliable");
    }
}
</script>
</body>
</html>
```

JavaScript Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers. For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse events:

Event Performed	Event Handler	Description
Click	Onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element

mouseout	onmouseout	When the cursor of the mouse leaves an element.
mouse down	onmousedown	When the mouse button is pressed over the element.
mouse up	on mouse up	When the mouse button is released over the element.
mousemove	On mousemove	When the mouse movement takes place.

Keyboard events:

Event Performed	Event Handler	Description
key down & keyup	onkeydown & On keyup	When the user press and then release the key

Form events:

Event Performed	Event Handler	Description
focus	Onfocus	When the user focuses on an element
Submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
Change	Onchange	When the user modifies or changes the value of a form element

Window/Document events

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page

unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Click Event

```
<html>
<head>Javascript Events </head>
<body>
<script language="Javascript" type="text/Javascript">
<!--
    function clickevent()
    {
        document.Write("This is Java Tpoint");
    }
    //-->
  </script>
  <form>
  <input type="button" onclick="clickevent()" value=
        "Who's this?"/>
  </form>
  </body>
  </html>
```

MouseOver Event

```
<html>
<head>
<h1> JavaScript Events </h1>
</head>
<body>
<script language="Javascript" type="text/Javascript">
  <!--
```

```
function mouseoverevent()
{
  alert("This is JavaTpoint");
}
//-->
</script>
<p onmouseover="mouseoverevent()">Keep cursor over me</p>
</body>
</html>
```

Focus Event

```
<html>
<head>Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" Onfocus="focusevent()"/>
<Script>
<!--
  function focusevent()
  {
   document.getElementById("input1").style.background="aqua";
  }
//-->
</script>
</body>
</html>
```

Keydown Event

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
```

```
<input type = "text" id = "input1" onkeydown =
      "keydownevent()" />
<Script >
<!--
   function keydownevent()
   {
      document.getElement By Id("input 1");
       alert ("Pressed a key ");
    }
//-->
</script>
</body>
</html>
```

JavaScript add EventListener()

The addEventListener() method is used to attach an event handler to a particular element. It doesnot override the existing event handlers. Events are said to be an essential part of the JavaScript. A Webpage responds according to the event that occurred. Events can be user-generated or generated by API'S. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

```
element.addEventListener(event, function, useCapture);
```

Parameter Values

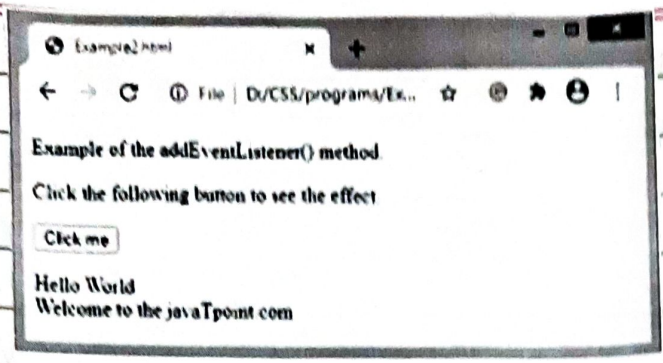
event: It is a required parameter. It can be defined as a string that specifies the event's name.

function: It is also a required parameter. It is a Java-script function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are true and false. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is false.

Example

```
<!DOCTYPE html>
<html>
<body>
 <P>Example of the addEventListener() method.</P>
 <P> Click the following button to see the effect.</P>
 <button id = "btn">click me </button>
 <p id = "para"></p>
 <script>
  document.getElementBy Id("btn").addEventListner("Click",fun);
  function fun() {
  document.getElementById("para").innerHTML = " Hello.
  World " + <br>" + "Welcome to the javaTpoint.com";
 }
 </script>
 </body>
 </html>
```



Event Bubbling or Event Capturing

Bubbling, the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first and then the outermost element's event will be handled.

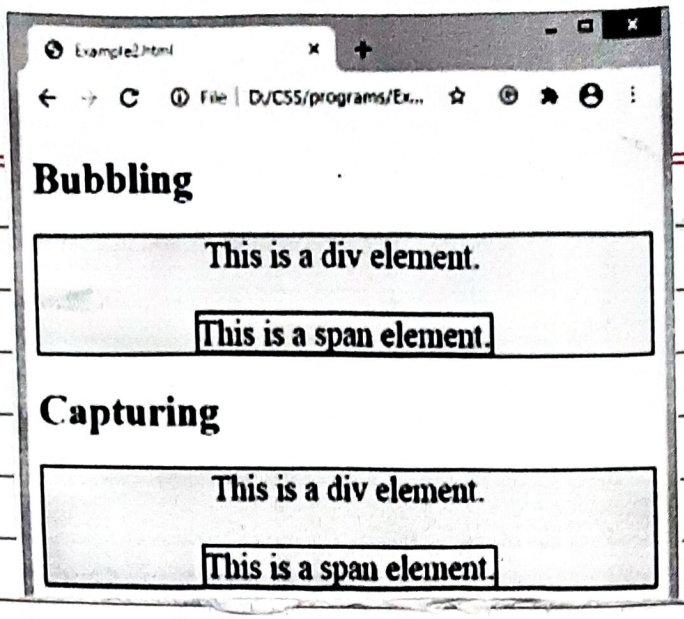
In Capturing the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

addEventListener(event, function, useCapture);

Example

```
<! DOCTYPE html>
<html>
<head>
<Style>
 div {
 background-color: lightblue;
 border: 2px solid red;
 font-size: 25px;
 text-align: center
 }
  span {
```

```
border: 2px solid blue;
}
</style>
</head>
<body>
<h1> Bubbling</h1>
<div id = "d1">
This is a div element.
<br><br>
<span id ="s1">This is a span element.</span>
</div>
<h1> Capturing </h1>
<div id = "d2">This is a div element.
<br><br>
<span id = "s2">This is a span element.</span>
</div>
<script>
document.getElementById("d1").addEventListener("dblclick",
function(){alert("You have double clicked on div element")},
false);
document.getElementById("s1").addEventListener("dblclick",
function(){alert("you have double clicked on span
element")}, false);
document.getElementById("d2").addEventListener("dblclick",
function(){alert('you have double clicked on div element')},
true);
document.getElementById("s2").addEventListener("dblclick",
function(){alert('you have double clicked on span
element')}, true);
</script>
</body>
</html>
```



JavaScript Onclick event

This event can be dynamically added to any element. It supports all HTML elements except <html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>, <br>, <meta>, and <param>. It means we cannot apply the onclick event on the given tags.

In HTML, we can use the onclick attribute and assign a Javascript function to it. We can also use the JavaScript's addEventListener() method and pass a click event to it for greater flexibility.

In HTML

```
<element onclick = "fun()">
```

In JavaScript

```
Object.onclick = function() {myScript};
```

In JavaScript by using the addEventListener() method

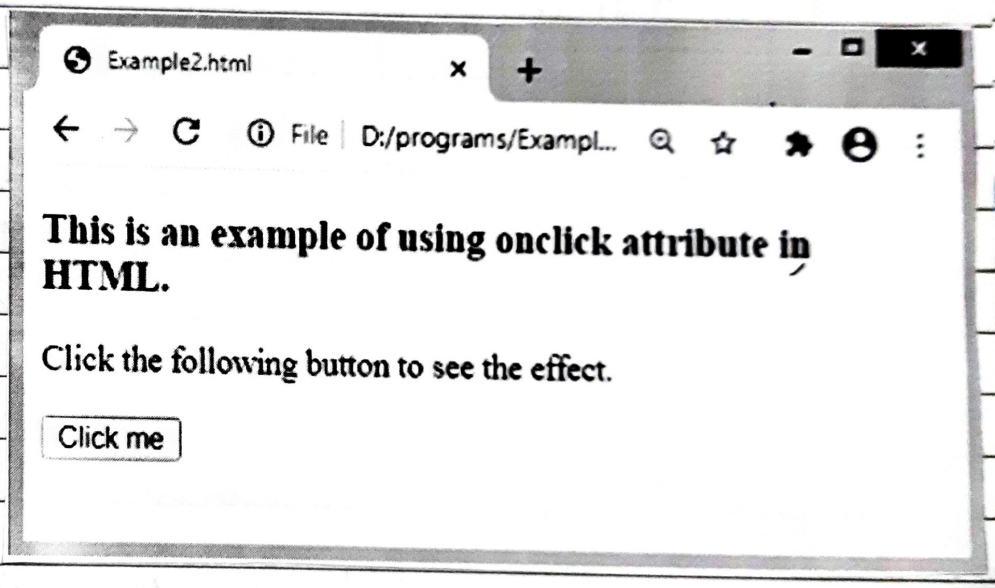
```
Object. addEventListener("click", myScript);
```

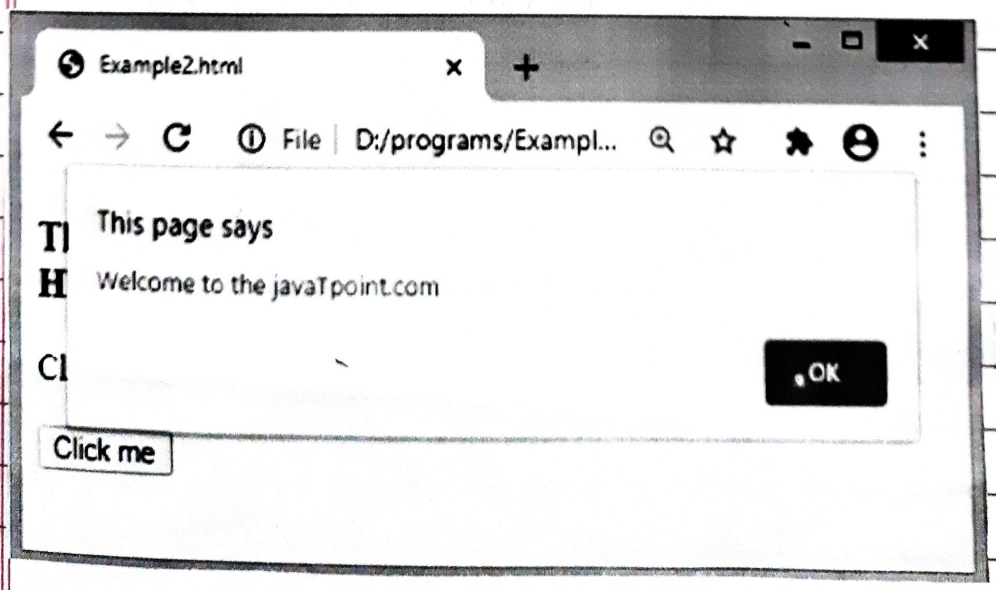
Example

```
<!DOCTYPE html>
<html>
```

```
<head >
<script>
function fun() {
alert ("Welcome to the javaTpoint.com");
}
</script>
</head>
<body>
<h3>This is an example of using onclick attribute in
    HTML.</h3>
<p>Click the following button to the see effect.</p>
<button onclick = "fun()">Click me</button>
</body>
</html>
```

Output





Example - Using JavaScript

```
<!DOCTYPE html>
<html>
<head>
 <title> onclick event </title>
 </head>
 <body>
  <h3> This is an example of using onclick event. </h3>
  <p> Click the following text to see the effect. </p>
  <p id = "Para"> Click me </p>
  <script>
   document.getElementById ("para").Onclick =
        function() {
    fun()
    };
    function fun() {
   document.getElementById("para").inner HTML =
        "Welcome to the javaTpoint. com";
   document.getElementById("para").style.Color="blue";
   document. get Element By Id("para"). style.
          backgroundcolor = "yellow";
   document.getElement By Id("para"). Style.fontSize=
          "25px";
   document.getElementById("para").style.border=
          "4px solid red";
  }
  </script>
  </body>
  </html>
```

JavaScript dblclick event

The dblclick event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's addEventListener() method to fire the double click event.

In HTML

We can use the ondblclick attribute to create a double click event.

In HTML

```
<element on dblclick = "fun()">
```

In JavaScript

```
object.ondblclick = function() {myScript};
```

In JavaScript by using the addEventlistener() method

```
object.addEventListener("dblclick", myScript);
```

Example - Using ondblclick attribute in HTML

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id = "heading" ondblcick = "fun()">Hello World :):)</h1>
<h2 > Double click the text "Hello World" to see the
    effect. </h2>
<P> This is an example of using the <b> ondblclick
    </b> attribute. </P>
<script>
```

```
function fun() {
document.getElementById("heading").innerHTML =
    "Welcome to the javaTpoint.com";
}
</script>
</body>
</html>
```

Example - Using JavaScript

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id = "heading">Hello World :):)</h1>
<h2> Double Click the text "Hello World" to see
    the effect. </h2>
<P> This is an example of Creating the double
    Click event using JavaScript.</p>
<Script>
document.getElementById("heading").ondblclick =
  function() { fun() };
function fun() {
document.getElementById("heading").innerHTML =
    "Welcome to the javaTpoint.com";
}
</script>
</body>
</html>
```

Example-Using JavaScript's addEventListener() method

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h1 id = "heading">Hello World :):) </h1>
<h2> Double Click the text "Hello World" to see the
    effect </h2>
<p> This is an example of Creating the double
 Click event using the <b> addEventListener() method </b>
<script>
document.getElementBy Id("heading") addEventListener
        ("dblclick", fun);
function fun() {
document.getElementBy Id ("heading"). innerHTML =
        "Welcome to the javaTpoint. com";
}
</script>
</body>
</html>
```

JavaScript onload

In HTML, the Onload attribute is generally used with the <body> element to execute a script once the content (including CSS files, images, scripts, etc.) of the Webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

```
Window.onlolad = fun()
```

Example

```
<!DOCTYPE html>
<html>
<head>
<meta charset = "utf-8">
<title> Window.onload()</title>
<style type = "text/css">
# bg {
width : 200 Px;
height : 200 Px;
border: 4 px solid blue;
}
</style>
<script type = " text/javascript ">
Window.onload = function(){
document.getElementById("bg").style.background
        Color = "red";
document.getElementById("bg").Style.Width = " 300 Px";
document.getElementById("bg")style.height = "300 px";
}
</script>
</head>
<body>
<h2> This is an example of Window.Onload()</h2>
<div id = "bg" ></div>
<body>
<html>
```

JavaScript onresize event

In HTML, we can use the onresize attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a resize event to it for greater flexibility.

In HTML

```
<element onresize = "fun()">
```

In JavaScript

```
Object.onresize = function() { my Script };
```

In JavaScript by using the addEventListener() method

```
Object.addEventListener("resize", my script);
```

Example

```
<!DOCTYPE html>
<html>
<head>
<Script>
var i = 0;
function fun() {
var res = "Width=" + Window.outerWidth+"<br>"+"Height = "
          + Window.outerHeight;
document.getElementById("para").innerHTML = res;
var res1 = i += 1;
document.getElementById("S1")innerHTML=res1;
}
</script>
</head>
<body onresize="fun()">
<h3>This is an example of using onresize attribute. </h3>
```

```
<p> Try to resize the browser's Window to See
    the effect. </p>
 <p id = "para"></p>
 <p> You have resized the Window <span id =
    "S1" >0 </ span >times. </p>
    </ body >
    </ html >
```

Example - Using JavaScript

```
<! DOCTYPE html >
< html >
< head >
 </ head >
  < body >
   <h3> This is an example of using JavaScripts
        onresize event. </h3>
   <p > Try to resize the browser's Window to
        See the effect. </p>
   <p id = " para "> </p>
   <p> You have resized the Window <span id=
    "S1" >0 </ span > times. </p>
   <script >
   document.getElementsByTagName ("BODY")[0].
     Onresize = function() { fun()};
   Var i =0;
   function fun() {
   Var res = "Width=" + window.outerWidth + "<br>"+
    "Height=" + Window.OuterHeight;
   document.getElement.ById ("para").innerHTML=res;
    Varres1 = i + = 1;
   document.get Element By Id ('S1'). innerHTML=res1;
  }
```

```
</script>
</body>
</html>
```

Example: Using addEventListener() method

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h3> This is an example of using JavaScript's addEventListener
        ()method. </h3>
<p> Try to resize the browser's window to see the
        effect. </p>
<p id = "para"></p>
<p> you have resized the window <span id ="s1">0
        </span > times. </p>
<script>
window.addEventListener("resize", fun);
var i = 0;
function fun() {
var res = "Width = " + window.outerWidth + "<br>" +
    "Height = " + window.outerHeight;
document.getElementById("para").innerHTML = res;
var res1 = i += 1;
document.getElementById("s1").innerHTML = res1;
}
</script>
</body>
</html>
```

Exception Handling in JavaScript

In programming, exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of the code/program. For handling the code, various handlers are used that process the exception and execute the code.

In exception handling
A throw statement is used to raise an exception. It means when an abnormal condition occurs, an exception is thrown using throw.
The thrown exception is handled by wrapping the code into the try block statements will get executed. Thus, in a programming language, there can be different types of errors which may disturb the proper execution of the program.

Types Of Errors

While coding, there can be three types of errors in the code:

1. Syntax Error :
When a user makes a mistake in the pre-defined syntax of a programming language, a syntax error may appear.

2. Runtime Error:
When an error occurs during the execution of the program, such an error is known as Runtime error. The codes which create runtime errors

are known as Exceptions. Thus, exception handlers are used for handling runtime errors.

3. Logical Error:
An error which occurs when there is any logical mistake in the program that may not produce the desired output, and may not produce the desired output, and may terminate abnormally. Such an error is known as Logical error.

Error Object

When a runtime error occurs, it creates and throws an Error object. Such an Object can be used as a base for the user-defined exceptions too. An error object has two properties:

1. name:
This is an Object property that sets or returns an error name.

2. message:
This property returns an error message in the string form.

Although Error is a generic constructor, there are following standard built-in error types or error constructors beside it:

1. EvalError:
It creates an instance for the error that occurred in the eval(), which is a global function used for evaluating the JS string code.

2. InternalError:
It creates an instance when the JS engine throws an internal error.

3. Range Error:
It creates an instance for the error that occurs when a numeric variable or parameter is out of its valid range.

4. ReferenceError:
It creates an instance for the error that occurs when an invalid reference is de-referenced.

5. SyntaxError:
An instance is created for the syntax error that may occur while parsing the eval().

6. Type Error:
When a variable is not a valid type, an instance is created for such an error.

7. URIError:
An instance is created for the error that occurs when invalid parameters are passed in encodeURI() or decodeURI().

JavaScript try... Catach

try{}statement:
Here, the code which needs possible error testing is kept within the try block. In case any error occur, it passes to the catch{}block for taking suitable actions and handle the error. Otherwise, it executes the code written within.

Catch{} statement:

The block handles the error of the code by executing the set of statements written within the block. This block contains either the user-defined exception handler or the buit-in handler. This block executes only when any error-prone code needs to be handled in the try block. otherwise, the catch block is skipped.

try.... catch example

```
<html>
<head> Exception Handling </br></head>
<body>
<Script>
try{
Var a = ["34", "32", "5", "31", "24", "44", "67"];//a is an array
document.Write(a); //displays elements of a
document.Write(b); //b is undefined but still trying to fetch
its Value. Thus Catch block Will be invoked
}Catch(e){
alert("There is error Which Shows"+e.message);//Handling
    error
}
</Script>
</body>
</html>
```

Throw Statement

Throw statements are used for throwing user-defined errors. User can define and throw their own custom errors. When throw statements is executed, the statements present after it will not execute. The control

Will directly pass to the catch block.

throw example with try....catch

```
<html>
<head> Exception Handling </head>
<body>
<script>
try {
    throw new Error ('This is the throw keyword');//
     user-defined throw statement
 }
 Catch (e) {
  document. Write (e.message); //This Will generate
     an error message
 }
 </script>
 </ body>
 </ html>
```

try....catach...finally statements

Finally is an optional block of statements which is executed after the execution of try and catch statements. Finally block does not hold for the exception to be thrown. Any exception is thrown or not, finally block code, if present execute. It does not care for the output too.

try... Catch... finally example

```
<html>
<head> Exception Handling </head>
<body>
<script>
```

```
try {
Var a = 2;
if (a==2)
document.Write("Ok")
}
Catch(Error){
document.Write("Error found"+e.message);
}
finally{
document.Write("Value of a is 2")
}
</script>
</body>
</html>
```

JavaScript this keyword

The this keyword is a reference variable that refers to the current object. Here, we will learn about this keyword with help of different examples.

Example

```
<Script>
Var address =
{
Company:"Javatpoint",
City:"Noida",
State: "UP",
full Address:function()
{
return this.company+" "+this.City+" "+this.state;
}
```

```
};
Var fetch = address.fullAddress();
 document.writeln(fetch);
 </script>
```

Output:
Javatpoint Noida UP

The following ways can be used to know which object is referred by this keyword.

Global Context

In global context, variables are declared outside the function. Here, this keyword refers to the window object.

```
<script>
Var website = "Javatpoint";
function web()
{
document.write(this.website)
}
web();
</script>
```

The call() and apply() method

The call() and apply() method allows us to write a method that can be used on different objects.

```
<script>
Var emp_address = {
   fullAddress: function() {
      return this.company +" " + this.city +" " +this.state;
   }
}
```

```
Var address = {
    Company : "Javatpoint",
    City : " Noida",
    State: " UP"
}
document.Writeln(emp_address.fullAddress.call(address));
document.Writeln(emp_address.fullAddress.apply(address));</script>
```

The bind() Method

The bind()method was introduced in ECMAScript 5. It creates a new function whose this keyword refers to the provided value, with a given sequence of arguments.

```
<Script>
Var lang = " Java";
function lang_name(call)
{
  Call();
};
Var Obj = {
   lang."JavaScript";
   language.function()
   {
   document.Writeln(this.lang+ "is a popular programming language");
   }
};
lang_name(obj.language);
lang_name(obj.languge.bind(obj));
</Script>
```

JavaScript Debugging

Sometimes a code may contain mistakes. Being a scripting language JavaScript didn't show any error message in a browser. But

these mistakes can affect the output.

JavaScript Debugging Example

Using console.log()method

The console.log() method displays the result in the console of the browser. If there is any mistake in the code, it generates the error message.

Example

```
<script>
 x = 10;
 Y = 15;
Z = x+Y;
 Console.log (z);
 Console.log(a);//a is not intialized
</script>
```

Using debugger keyword

In debugging, generally we set breakpoints to examine each line of code step by step. There is no requirement to perform this task manually in JavaScript.

```
<script>
 x = 10;
 Y = 15;
 Z = x+y;
 debugger;
 document.Write(Z);
 document.Write (a);
 </script>
```

JavaScript Hoisting

Hoisting is a mechanism in JavaScript that moves the

declaration of variables and functions at the top. so, in JavaScript we can use variables and functions before declaring them.

JavaScript Hoisting Example

Here, we will use the variable and function before declaring them.

JavaScript Variable Holsting

```
<script>
 x = 10;
document.Write(x);
var x;
</script>
```

JavaScript Function Hoisting

```
<script>
document.Write (sum (10,20));
function sum(a,b)
{
return a+b;
}
</script>
```

JavaScript Strict Mode

Being a scripting language, sometimes the JavaScript code displays the correct result even it has some errors. To overcome this problem we can use the Javascript Strict mode.

The JavaScript provides "use strict"; expression to enable the strict mode. If there is any silent error or mistake in the code, it throws an error.

Example
To print sum of two numbers.

```
<script>
console.log (sum(10,20));
function sum (a,a)
{
"use strict";
return a+a;
}
</script>
```

JavaScript Promise

Promises in real-life express a trust between two or more persons and an assurance that a particular thing will surely happen. In javascript, a Promise is an object which ensures to produce a single value in the future. Promise in javascript is used for managing and tacking asynchronous operations.

Terminology of Promise

A promise can be present in one of the following states:

1. pending:
   The pending promise is neither rejected nor fulfilled yet.

2. fulfilled:
   The related promise action is fulfilled successfully

3. rejected:
   The related promise action is failed to be fulfilled

4. settled:
Either the action is fulfilled or rejected.
Thus, a promise represents the Completion of an asynchornous operation With its result. It can be either successful Completion of the promise, or its failure, but eventually Completed. Promise uses a then() Which is executed only after the completion of the promise resolve.

Promises of Promise

A JavaScript Promise promises that:

1. Unless the current execution of the js event loop is not completed (success or failure) Callbacks Will never be called before it.
2. Even if the Callbacks With then() are present, but they Will be called Only after the execution of the asynchronous operations completely.
3. When multiple callbacks can be included by invoking then() many times, each of them Will be executed in a chain, ie., one after the other, following the sequence in Which they Were inserted.

Methods in Promise

The functions of Promise are executable almost on every trending Web browsers such as chrome, Mozilla, Opera, etc.

1. Promise.resolve(promise)
This method returns promise only if Promis.constructor == Promise.

2. Promise.resolve(thenable)
Makes a new promise from thenable Containing then().

3. Promise.resolve (obj)
Makes a promise resolved for an object.

4. Promise.reject (obj)
Makes a promise rejection for the object.

5. Promise.all(array)
Makes a promise resolved when each item in an array is fulfilled or objects when items in the array are not fulfilled.

6. Promise.race(array)
If any item in the array is fulfilled as soon it resolves the promise, or if any item is rejected as soon it rejects the promise.

Constructor in Promise

new Promise(function(resolve,reject){});
Here, resolve(thenable) denotes that the promise will be resolved with then()
Resolve(obj) denotes promise will be fulfilled with the object.
Reject (Obj) denotes promise rejected with the object.

Promise Implementation

```
<html>
<head>
<h2> Javascript Promise </h2>
</br> </head>
<body>
<script>
```

```
Var p = new Promise (function(resolve, reject) {
Var x = 2+3;
if (x == 5)
   resolve ("executed and resolved successfully");
else
  reject("rejected");
});
P. then(function (from Resolve) {
document. Write("Promise is" + from resolve);
}). catch (function( from Reject) {
document. Write("Promise is" + from Reject);
});
</script>
</body>
</html>
```

In the above Promise implementation the Promise constructor takes an argument that callbacks the function. This Callback function takes two arguments, ie.,

1. Resolve:
When the promise is executed successfully, the resolve argument is invoked, which provides the result.

2. Reject:
When the promise is rejected, the reject argument is invoked, which results in an error.

It means either resolve is called or reject is called. Here, then() has taken one argument which will execute, if the promise is resolved. Otherwise, catch() will be called with the rejection of the promise.

Advantages of using Promises

1. A better option to deal with asynchronous operations.
2. Provides easy error handling and better code readability.

JavaScript Compare dates

In the previous section, we discussed the date methods as well the constructors. Here, with the help of those methods, we will learn to compare dates.
Basically there are different ways to which we can compare dates, such as;

1. Comparing two dates with one another.
2. Comparing date with time.
3. Comparing dates using getTime()

Comparing two dates with one another

Example

```
<html>
<head> Comparing Dates</br></head>
<body>
<Script>
function compare()
{
Var d1 = new Date ('2020-01-23');//YYYY-mm-dd
Var d2 = new Date ('2020-01-21');//YYYY-mm-dd
if ( d1 > d2)
{
document.write ("True, First date is greater than Second
                date");
}
```

```
else if (d1 < d2)
{
document.Write("False,Second date is smaller than the first");
}
else
{
document. Write("Both date are same and equal");
}
}
Compare(); // invoking compare()
</script>
</body>
</html>
```

Comparing date With time

Example

Comparing different dates with different things

```
<html>
<head> Comparing Date and time </br></head>
<body>
<Script>
Var d1 = new Date("Apr 17, 2019, 12:10:10"); //mm dd.yyyyhh.mm:ss
Var d2 = new Date ("Dec 1, 2019 12:10:30"); //mm. ddyyyyhh.mm.ss
if (d1 > d2)
{
document.Write (false, d1 date and time is smaller than
                d2 date and time");
}
else if (d1 < d2)
{
document. Write ("True, d2 is greater than in terms of
                both time and date");
```

```
}
else
document.Write("Both date and time are same
                and equal");
}
</script>
</body>
</html>
```

Comparing date with getTime()

Example

Comparing current date and time with a given date and time.

```
<html>
<head> comparing Dates</br></head>
<body>
 <script>
Var d1 = new Date("2019-10-10, 10:10:10");//yyyy-mm-dd
                                hh:mm:ss
Var currentdate = new Date();//fetch the current date
                                value
if(d1.getTime()< currentdate.getTime())
{
document.Write("True. currentdate and time are greater
                than d1");
}
else if(d1.getTime()> currentdate.getTime())
{
document.Write("False");
}
else
```

```
{
document.Write("True, equal");
}
</script>
</body>
</html>
```

JavaScript array.length property

The length property can also be used to set the number of elements in an array. We have to use the assignment operator in conjuction with the length property to set an array's length.

The array.length property in JavaScript is same as the array.size() method in jQuery. In JavaScript, it is invalid to use array.size() method so, we use array.length property to calculate the size of an array.

array.length

array.length = number

Example

```
<html>
<head>
<title> array.length </title>
</head>
<body>
<h3> Here, we are finding the length of an array. </h3>
<script>
Var arr = new Array (100,200, 300, 400, 500, 600);
document.write("The elements of array are" +arr);
```

```
document.Write("<br> The length of the array is:" +
                    arr.length);
</script>
</body>
</html>
```

JavaScript alert()

The alert() method displays an alert box with a message and an ok button.
The alert() method is used when you want information to come through to the user.

Note
The alert box takes the focus away from the current window, and forces the user to read the message.

```
alert(message)
```

Example

```
<html>
  <head>
   <script type = "text/javascript">
      function fun() {
      alert ("This is an alert dialog box");
    }
   </script>
  </head>
  <body>
   <p> Click the following button to see the effect</p>
   <form>
    <input type = "button" value = "Click me" onclick=
                    "fun();" />
```

```
</form>
</body>
</html>
```

JavaScript eval() function

The eval()function in JavaScript is used to evaluate the expression. It is JavaScirpt's global function, which evaluates the specified string as JavaScript code and executes it. The parameter of the eval()function is a string. If the parameter represents the statements, eval() evaluates the statements. If the parameter is an expression, eval() evaluates the expression. If the parameter of eval() is not a string, the function returns the parameter unchanged.

```
eval(String)
```

Example

```
<html>
<head>
<script>
var a=10, b=20, c=30, sum, mul, sub;
sum = eval("a+b+c");
mul = eval("a*b*c");
sub = eval("a-b");
document.write(sum+"<br>");
document.write(mul+"<br>");
document.write(sub);
</script>
</head>
<body>
</body>
```

```
</html>
```

Output

```
60
6000
-10
```

JavaScript closest()

The closest() method in JavaScript is used to retrieve the closest ancestor, or parent of the element matches the selectors. If there is no ancestor found, the method returns null.
This method traverses the element and its parents in the document tree, and the traversing continues until the first node is found that matches the provided selector string.

```
targetElement.closest(selectors);
```

Example

```
<!DOCTYPE html>
 <html>
  <head>
  </head>
  <body>
  <div id = "div 1"> This is the first div element.
    <h3 id = "h" >This is a heading inside the div.</h3>
  <divid = "div2"> This is the div inside the div element
  <divid = "div3> This is the div element inside the
              second by div element.</div>
   </div>
```

```
</div>
  <script>
var Val1 = document.getElementById("div3");
  var01 = Val1.Closest ("# div1");
  var02 = Val1 Closest ("div div");
  var03 = Val1 closest ("div > div");
  var04 = Val1 closest (":not(# div3)");
    Console.log (01);
    Console.log (02);
    Console.log (03);
    Console.log (04);
      </script>
    </body>
    </html>
```

JavaScript continue statement

The continue statment in JavaScript is used to jumps over an iteration of the loop. Unlike the break statement, the continue statement breaks the current iteration and continues the execution of next iteration of the loop. It can be used in for loop, and do-while loop. When it is used in a while loop, then it jumps back to the condition. It is used in for loop, the flow moves to the update expression.

When we apply the continue statement, the program's flow immediately moves the conditional expression, and if the condition is true, then the next iteration will be started; otherwise, the control exits the loop.

```
Continue;
OR
Continue[label]; // Using the label reference
```

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <h1> Example of the Continue statement in JavaScript</h1>
    <h3> Here, you can see that "a==4" is skipped.</h3>
    <p id = "para">
    </p>
    <script>
    Var res = " ";
Var a;
for (a=1; a<=7; a++){
  if (a==4){
  Continue;
 }
 res+= "The Value of a is:" +a +"<br>";
}
 document.getElementById("para").innerHTML=res;
        </script>
    </body>
    </html>
```

JavaScript getAttribute()method

The getAttribute() method is used to get the value of an attribute of the particular element. If the attribute exists, it returns the string representing the value of the corresponding attribute. If the corresponding attribute does not exist, it will return an empty string or null.

element.getAttribute(attributename)

Example

```
<!DOCTYPE html>
<html>
<head>
<title>
The getAttribute Method
</title>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
</h1>
<h2>
Example of the getAttribute() Method
</h2>
<div id = "div1" style = "background-color: yellow; front-size:
25px; color: Red; border: 2px solid red;">
This is first div element.
</div>
<br>
<div id = "div2" style = "background-color: lightblue;
size 25px; color: blue; border : 2px solid blue;">
This is second div element.
</div>
<br>
<button onclick = "fun()">
Click me!
</button>
<p id = "p"></p>
<p id = 'p1"></p>
```

```
<Script>
function fun() {
Var val = document.getElementById("div1").getAttribute
                ("Style");
document.getElementById("p") innerHTML = Val;
Var Val1 = document.getElementById ("div2").
            getAttribute("style");
document.getElementById("p1") innerHTML = Val1;
}
</script>
<lbody>
</html>
```

JavaScript hide elements

In JavaScript, we can hide the elements using the style.display or by using the style.visibility. The visibility property in JavaScript is also used to hide and element. The difference between the style.display and style.visibility is when using visibility: hidden, the tag is not visible, but space is allocated. Using display: none, the tag is also not visible but there is no space allocated on the page.
In HTML, we can use the hidden attribute to hide the specified element. When the hidden attribute in HTML sets to true, the element is hidden, or when the value is false, the element is visible.

Using style.hidden

```
document.getElementById ("element") style.display = "none";
```

Using Style.visibility

```
document.getElement By Id ("element") style.visibility = "none";
```

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>
style.display
</title>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
</h1>
<h2>
<div id="div" style="background-color:yellow; font-size:25px;
color:red; border:2 px solid red;">
This is the div element.
</div>
<p id="p">This is a paragraph element.</p>
<button onclick="fun()" id="btn">
click me!
</button>
<script>
function fun() {
document.getElementById("div").style.display="none"
document.getElementById("p").style.display="none"
}
</script>
</body>
</html>
```

JavaScript prompt() dialog box

The prompt() method in JavaScript is used to display a prompt

box that prompts the user for the input. It is generally used to take the input from the user before entering the page. It can be written without using the window perfix. When the prompt box pops up, we have to click "Ok" or "Cancel" to proceed.

The box is displayed using the prompt() method, which takes two arguments: The first argument is the label which displays in the text box, and the second argument is the default string, which displays in the textbox. The prompt box consists of two buttons, Ok and Cancel. It returns null or the string entered by the user. When the user clicks "Ok", the box returns the input value. otherwise, it returns null on clicking "Cancel".

Prompt(message, default)

Example

```
<html>
<head>
<Script type = "text/javascript">
function fun() {
Prompt("This is a prompt box", "Hello world");
}
</script>
</head>
<body>
<p> Click the following button to see the effect</p>.
<form>
<input type = "button" value = "Click me" onclick = "fun()"/>
</form>
</body>
</html>
```

Output

After the execution of the above code and clicking the clickme button, the output will be -

JavaScript removeAttribute() method

The method is used to remove the specified attribute from the element. It is different from the removeAttributeNode() method. The removeAttributeNode() method removes the particular Attr object, but the removeAttribute() method removes the attribute with the specified name.

element.removeAttribute(attributename)

Example

```
<!DOCTYPE html>
<html>
<head>
<title>
The removeAttribute Method
</title>
<style>
jtp{
color: red;
backgroud-color: yellow;
}
</style>
</head>
<body>
<h1>
Welcome to the javaTpoint.com
```

```
</h1>
<h2>
Example of the removeAttribute Method
</h2>
<p id = "para" class = "jtp">
This is a paragraph element.
</p>
<p id = "para1" class = "jtp">
This is second paragraph element
</p>
<button onclick = "fun()">
Click me!
</button>
<script>
function fun() {
document.getElementById("para").removeAttribute("class");
document.getElementById("para1").removeAttribute("class");
}
</script>
</body>
</html>
```

JavaScript reset

In HTML, we can use the reset button to reset the form. In this article, we are discussing how to reset the form using JavaScript.

In JavaScript, the reset() method does the same thing as the HTML reset button. It is used to clear all the values of the form elements. It can be used to set the valus to defalult. It does not require any parameter values and also does not return any value.

formElement.reset()

Example

```
<!DOCTYPE html>
<html>
<head>
<title>reset()method</title>
</head>
<body style="text-align:center;">
<div style="background:pink;">
  <font color="red" size="6px">
   <b> Example of the reset() method</b>
  </font>
  </div>
   <div style="background: lightblue;">
   <form id="myForm" action="#" style="font-size:20px;">
   <p>First Name:<input type="text" id="fname"/></p>
   <p>Last Name:<input type="text" id="lname"/></p>
   <p>E-mail Id:<input type="email" id="email"/></p>
   <p>Age:  <input type="number" id="age"/></p>
   <input type="submit">
   <input type="button" value="Reset data" onclick="fun()"/>
   </form>
   </div>
 <script>
  function fun(){
   document.getElementById("myForm").reset();
 }
 </script>
 </body>
 </html>
```

JavaScript return

The return statement is used to return a particular Value from the function to the function caller. The function will stop executing when the return statement is called. The return statement should be the last statement in a function because the code after the return statement will be unrechable.
We can return primitive values and Object types by using the return statement.
We can also return multiple values using the return statement. It cannot be done directly. We have to use an Array or Object to return multiple values from a function.

return expression;

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<h2> Welcome to the javaTpoint.com </h2>
<h3> Example of the Javascript's return Statement </h3>
    <script>
Vares = fun(12,30);
function fun(x,y)
{
return x * y;
}
document.write(res);
    </script>
```

```
</body>
</html>
```

JavaScript String split()

As the name implies, the split() method in JavaScript splits the string into the array of substrings, puts these sub-strings into an array, and returns the new array. It does not change the original string.

When the string is empty, rather than returning an empty array, the split() method returns the array with an empty string. The empty array is returned when both string and separator are empty strings

String.split(separator, limit)

Example

```
<html>
<head>
<script>
Var Str = 'Welcome to the javaTpoint.com'
Var arr = Str.split(" ", 3);
document.Write(arr);
</script>
</head>
<body>
</body>
</html>
```

Output

Welcome, to, the

JavaScript typeof operator

The JavaScript typeof operator is used to return a string that represents the type of JavaScript for a given value. It returns the data type of the operand In the form of a string. The operand can be a literal or a data structure like a function, an object or a variable.

```
typeof operand
or
typeof(operand)
```

The possible returns values of the typeof operator are tabulated as follows:

Type of the operand	Result
Object	"Object"
number	"number"
string	"string"
boolean	"boolean"
function	"function"
undefined	"undefined"

Example

```
<html>
<head>
<script>
document.write(typeof(45)+"<br>"); //results: "number"
document.write(typeof(-90)+"<br>"); //results: "number"
document.write(typeof(0) +"<br>"); //results: "number"
document.write(typeof(22.6)+"<br>"); //results: "number"
document.write(typeof(Infinity)+"<br>"); //results: "number"
```

```
document.write(typeof(NaN));//results:"number". Although
        Nan is "Not-A-Number"
</script>
</head>
<body>
</body>
</html>
```

Output

After the execution of the above code, the output will be -

```
number
number
number
number
number
number
```

JavaScript ternary operator

During coding in any language, we use various ways to handle conditional situations. The common one is the use of if statement; instead of using the if statement, we can use the ternary operator in JavaScript. The ternary operator assigns a value to the variable based on a condition provided to it.

```
Var a = (condition)? expr1 : expr2;
```

Example

```
<!DOCTYPEhtml>
<head>
<Script>
```

```
let a = 358;
let val = (a % 2 == 0)?'Even Number': 'Odd Number';
alert(val);
</script>
</head>
<body>
<h1>Welcome to the javaTpoint.com</h1>
<h3>This is an example of ternary operator.</h3>
</body>
</html>
```

JavaScript reload() method

In Java Script, the reload() method is used to reload a webpage. It is similar to the refresh button of the browser. This method does not return any value.

```
location.reload()
```

Example

```
<!DOCTYPE html>
<html>
<head>
<title>
location.reload()method
</title>
<script>
function fun() {
location.reload();
}
</script>
</head>
<body>
```

```
<h1> Welcome to the JavaTpoint.com </h1>
<h2> This an example of location.reload()method</h2>
<p> Click the following 'Reload' button to see the effect
                                    </p>
<button onclick="fun()" >Reload</button>
</body>
</html>
```

JavaScript setAttribute()

The setAttribute() method is used to set or add an attribute to a particular element and provides a value to it. If the attribute already exists, It only set or changes the value of the attribute. So, we can also use the setAttribute() method to update the existing attribute's value. If the corresponding attribute does not exist, It will create a new attribute with the specified name and value. This method does not return any value. The attribute name automatically converts into lowercase when we use it on an HTML.

element.setAttribute(attributeName, attributeValue)

Example

```
<html>
<head>
<title> JavaScript setAttribute()method </title>
<Script>
function fun(){
document.getElementById("link").SetAttribute("href", "https:
                //WWW.javaTpoint.com/ ");
}
```

```
</script>
</head>
<body style = "text-align: center;" >
<h2> It is an example of adding an attribute
    using the setAttribute() method. </h2>
<a id = "link" > javaTpoint.com </a>
<p> Click the following button to see the effect. </p>
<button onclick = "funU" > Add attribute </button>
</body>
</html>
```

JavaScript setInterval() method

The setInterval() method in JavaScript is used to repeat a specified function at every given time-interval. It evaluates an expression or calls a function at given intervals. This method continues the calling of function until the Window is closed or the clearInterval() method is called. This method is returns a numeric value or a non-zero number that identifies the created timer.

Window.setInterval(function, milliseconds);

Example

```
<html>
<head>
<title> setInterval() method </title>
</head>
<body>
<h1> Hello world :) :) </h1>
<h3> This is an example of using the sentInterval() method
    </h3>
```

```
<p> Here, an alert dialog box displays on every three seconds.
</p>
<Script>
Var a;
a = sentInterval (fun, 3000);
function fun() {
alert ("Welcome to the javaTpoint.com")
} </script>
 </body>
 </html>
```

JavaScript setTimeout() method

The setTimeout() method in JavaScript, is used to execute a function after waiting for the specified time interval. This method returns a numeric value that represents the Id value of the timer.

We can use the clearTimeout() method to stop the timeout or to prevent the execution of the function specified in the setTimeout() method. The value returned by the setTimeout() method can be used as the argument of the clearTimeout() method to cancel the timer.

```
Window.setTimeout (function, milliseconds);
```

Example

```
<html>
<head>
<title> setTimeout() method </title>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the setTimeout() method </h3>
```

```
<p>Here, an alert dialog box will display after two
        seconds. </p>
<script>
var a;
a = setTimeout(fun, 2000);
function fun() {
alert("Welcome to the javaTpoint.com");
}
</script>
</body>
</html>
```

JavaScript String includes()

The JavaScript string includes() method is used to determine whether or not the specified substring is present in the given string. It is a case-sensitive method. It returns the Boolean value, either true or false. It returns true if the string contains the specified substring and returns false if not.
It does not change the value of the original string.

```
String.includes(searchValue, start);
```

Example

```
<DOCTYPE html>
<html>
<head>
</head>
<body>
<h1> Hello World :) :) </h1>
<h3> This is an example of using the JavaScript's
        String includes() method </h3>
```

```
<script>
let str = "Welcome to the javaTpoint.com";
document.Write("<b> The given string is: </b>", str);
document.Write ("<br>");
let res = str.includes('tO');
document.write("<b> The result is: </b>", res);
</script>
</body>
</html>
```

JavaScript String trim()

The trim() is a built-in string function in JavaScript, which is used to trim a string. This function removes the Whitespace from both the ends; ie, start and end of the string. As the trim() is a string method, so it is invoked by an instance of the string class. We have to create an instance of String class.

```
Str.trim()
```

Example

```
<html>
<body>
<script>
function func_trim() {
    //Original string with whitespace in beginning and end
  Var str = " javatpoint tutorial website ";
  //String trimmed using trim()
  Var trimmedstr = str.trim();
  document.write(trimmedstr);
}
func_trim();
```

```
</script>
</html>
</body>
```

Output
Javatpoint tutorial website

Javascript Setinterval

Javascript can be made to execute a block of code at specific intervals of time. These intervals are critically defined as time events. There are usually two methods for the same. They can be specifically used according to your requirements. Those two methods are:

1. SetInterval()
2. setTimeout()

Example Code:

```
function Display()
{
  Console.log("Hello JavaTpoint");
}
SentInterval(Display, 2000);
```

Code 2:

```
Var logic = setInterval(Time, 2000);
function Time (){
  Var x = new Date();
  Vary = x.toLocaleTimeString();
}
function FunctionStop(){
  ClearInterval (logic);
}
```

JavaScript print() method

A print() method is used to print the currently visible contents like a web page, text, image, etc, on the computer screen. When we use a print() method in JavaScript and execute the code, it opens a print dialog box that allows the user or programmer to select an appropriate option for printing the current content of the window.

Example

```
<html>
<head>
<title>
Use print() method in JavaScript
</title>
<script type = "text/javascript">
<!--
//-->
</script>
</head>
<body>
<h2> program to print the Current Content of the
        Window using print() method </h2>
<br> <br>
<p>
As the name suggests, the print() method is used to
print the Contents of the Current Window.
<form>
<!-
-When a user click on the print button, the onclick function
Calls the Window.print()method
->
<input type = "button" value = "Print" onclick = "window.print()" />
```

```
</form>
</body>
</html>
```

JavaScript TypedArray

The JavaScript TypedArray object illustrates an array like view of an underlying binary data buffer. There are many number of different global properties, whose values are TypedArray constructors for specific element types.

Types of TypedArray

Int8Array
- Size in bytes: 1
- Description: 8-bit two's compliment signed integer.
- Type: byte.
- Value Range: -128 to 127

Unit8Array
- Size In bytes: 1
- Description: 8-bit two's compliment signed octet.
- Type: Octet.
- Value Range: 0 to 255.

Unit8ClampedArray
- Size in bytes: 1
- Description: 8-bit unsigned integer(clamped) octet.
- Type: octet
- Value Range: 0 to 255.

Int16Array

- Size in bytes : 2
- Description : 16-bit two's complement signed integer.
- Type : short
- Value Range : -32768 to 32767.

Unit16Array
- Size in bytes : 2
- Description: 16- bit unsigned integer.
- Type: unsigned short.
- Value Range : 0 to 65535.

Int32Array
- Size in bytes : 4
- Description : 32-bit two's complement signed integer.
- Type : long.
- Value Range :-2147483648 to 2147483647.

Unit32Array
- Size in bytes : 4
- Description: 32-bit unsigned integer.
- Type : unsigned long.
- Value Range: 0 to 4294967295.

Float32Array
- Size in bytes : 4
- Description: 32- bit IEEE floating point number unrestricted float.
- Type : unrestricted float.
- Value Range: 1.2×10^{-38} to 3.4×10^{38}

Float 64Array
- Size in bytes : 8
- Description: 64-bit IEEE floating point number unrestricted double.

- Type: unrestricted double.
- Value Range: 5.4 x 10-324 to 1.8 x 10308

JavaScript Set Object

The JavaScript Set object is used to store the elements with unique values. The values can be of any type i.e. whether primitive values or object references.

new Set([iterable])

Parameter

interable- It represents an iterable object whose elements will be added to the new set.

Points to remember

- A set object uses the concept of keys internally.
- A set object cannot contain the duplicate values.
- A set object iterates its elements in insertion order.

JavaScript Set Methods

1. add()
   It adds the specified values to the Set Object.

2. Clear()
   It removes all the elements from the Set object.

3. delete()
   It deletes the specified element from the set object.

4. entries()

It returns an object of set iterator that contains an array of [value, value] for each element.

5. forEach()
It executes the specified function once for each value.

6. has()
It indicates whether the set object contains the specified value element.

7. Values()
It returns an object of set iterator that contains the values for each element.

JavaScript Map Object

The JavaScript Map object is used to map keys to values. It stores each element as key-value pair. It operates the elements such as search, update and delete on the basis of specified key.

new Map ([iterable])

Parameter

Iterable - It represents an array and other iterable object whose elements are in the form of key-value pair.

Points to remember

- A map object cannot contain the duplicate keys.
- A map object can contain the duplicate values.
- The key and value can be of any type (allows both object and primitive values).
- A map object iterates its elements in insertion order.

JavaScript Map Methods

1. Clear()
It removes all the elements from a Map object.

2. delete()
It deletes the specified element from a Map object.

3. entries()
It returns an object of map interator that contains the key-value pair of each element.

4. forEach()
It executes the specified function once for each key/value pair.

5. get()
It returns the value of specified key.

6. has()
It indicates whether the map object contains the specified key element.

7. Keys()
It returns an object of Map iterator that contains the keys for each element.

8. set()
It adds or updates the key-value pairs to Map object.

9. Values()
It returns an object of map iterator that contains the values for each element.

JavaScript WeakSet Object

The JavaScript WeakSet object is the type of collection that allows us to store weakly held objects. Unlike set, the weakset are the collections of objects only. It doesnt contain the arbitrary values.

new WeakSet([iterable])

Parameter

iterable - It represents the iterable object whose elements will be added to a new WeakSet.

Points to remember

- A WeakSet object contains unique objects only.
- In WeakSet, if there is no reference to a stored object, they are targeted to garbage collection.
- In WeakSet, the objects are not enumerable. so, It doesnt provide any method to get the specified objects.

JavaScripts WeakSet Methods

1. add()
It adds a new object to the end of WeakSet object.

2. delete()
It removes the specified object from the WeakSet object.

3. has()
It indicates whether the WeakSet object contains the specified object element.

JavaScript Closures

A closure can be defined as a JavaScript feature in which the inner function has access to the outer function variable. In JavaScript every time a closure in created with the creation of a function.
The closure has three scope chains listed as follows:

- Access to its own scope.
- Access to the variables of the outer function.
- Access to the global variables.

Example

```
<!DOCTYPE html>
<html>
<head>
<Script>
function fun()
{
Var a = 4;//'a' is the local variable, Created by the fun()
function innerfun()//the innerfun() is the inner function,
    Or a closure
{
return a;
}
return innerfun;
}
Var output = fun();
document.Write(output());
document.Write(" ");
document.Write(output());
</Script>
</head>
<body>
```

JavaScript date format

The JavaScript date object can be used to get a year, month and day. We can display a timer on the Web-page with the help of a JavaScript date object. There are many types of date formats in JavaScript: ISO Date, Short Date and Long Date. The format's of JavaScript's date are defined as follows:

ISO date

"2020-08-01" (The International Standard)

Short date

"01/08/2020"

Long date

"Aug 01 2020" or "01 Aug 2020"

ISO date

The ISO 8601 is the international standard for the times and dates, and the syntax (YYYY-MM-DD) of this standard is the preferred date format in JavaScript

Example

```
<!DOCTYPEhtml>
<html>
<head>
</head>
<body>
<div>
<h1>Welcome to the JavaTpoint.com</h1>
<h3>It is an example of JavaScript's ISO date</h3>
<P id = "para"></p>
```

```
</div>
<script>
let val = new Date ("2020-08-01");
document.getElementById("para").innerHTML=val;
</script>
</body>
</html>
```

We can write the ISO dates using the following syntaxes:

1. This is a complete date format using ISO date.
   let val = new Date ("2020-08-01");
   Sat Aug 01 2020 05:30:00 GMT +0530 (India Standard Time)

2. In this format, we specify only year and month (YYYY-MM) without day.
   let val = new Date ("2020-08");
   Sat Aug 01 2020 05:30:00 GMT+0530 (India standard Time)

3. In the third syntax, we only specify the year (YYYY) without month and day.
   let val = new Date ("2020")
   Wed Jan 01 2020 05:30:00 GMT+0530 (India standard Time)

4. Now, in the forth syntax, we specify the date with added hours, minutes and seconds. (YYYY-MM-DDTHH:MM:SSZ). In this format, the date and time are separated with the letter 'T' and the letter 'Z'. We get different results in different browsers if we remove these characters.
   let val = new Date ("2020-08-01T07:05:00Z");
   Sat Aug 01 2020 12:35:00 GMT+0530 (India Standard Time)

JavaScript Short Date

The "MM/DD/YYYY" is the format used to Write short dates. Now we understand the short date by using an example.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<div>
<h1> Welcome to the JavaTpoint.com </h1>
<h3> It is an example of JavaScript's Short date </h3>
</div>
<Script>
let val = new Date("08/01/2020");
document.Write(val);
</script>
</body>
</html>
```

JavaScript Long Date

The "MMM DD YYYY" is the format used to Write Long dates. The month and date can be written it any order. and it is allowed to Write a month in addreviated (Aug) form or in full (August)

Example

```
<!DOCTYPE html>
<html>
<head>
```

```
</head>
<body>
<div>
<h1> Welcome to the JavaTpoint.com</h1>
<h3> It is an example of JavaScript's Long date</h3>
</div>
<script>
let val= new Date ("Aug 01 2020");
document.Write(val);
</script>
</ body>
</ html>
```

JavaScript date parse() method

The parse() method in JavaScript is used to parse the specified date string and returns the number of milliseconds between the specified date and January 1, 1970. If the string does not have valid values or if it is not recognized, then the month method returns NaN.

The counting of milliseconds between two specified dates helps us to find the number of hours, days, months, etc. by doing easy calculations.

```
date.parse(datestring)
```

Example

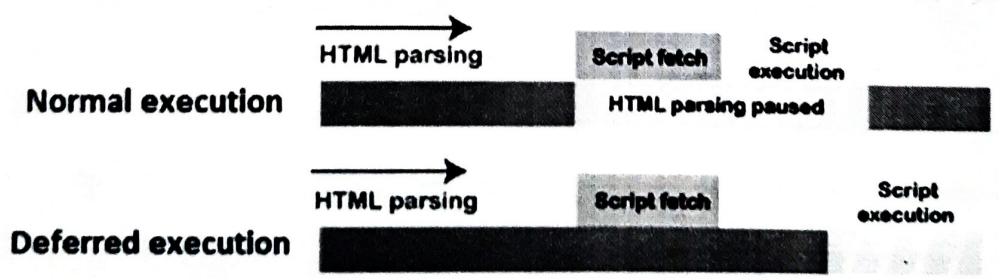
```
<html>
<head>
</head>
<body>
<h1> Hello World :) :) </h1>
```

```
<p>Here, we are finding the number of milliseconds between
the given date and midnight of JavaScript
<script>
var d1 = "June 19, 2020";
Var m1 = Date.parse(d1);
document.write("The number of milliseconds between <b>"
  + d1 + "</b> and <b> January 1, 1970 </b> is: <b>" + m1 +
                "</b>");
</script>
</body>
</html>
```

JavaScript defer

The defer is Boolean value, used to indicate that script is executed after the document has been parsed. It works only with externals scripts.(i.e.; works only when we are specifying the src attribute in <script> tag). It declares that the script will not create any content. So the browser can continue the parsing of the rest of the page. The <script> with the defer attribute does not block the page.

We can understand the use of defer attribute in the following image:



This attribute tells the browser to execute the <script> file when the entire HTML document gets fully parsed. sometimes, the application consumes more memory by adding the <script> tag in the HTML head section, and it also causes performance issues. To improve the performance, we can add the defer attribute in the <script> tag.

<script defer>

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="myscript.js" defer>
</script>
</head>
<body>
<div>
<div>
<h1> javaTpoint.com </h1>
<h3> This is an example of defer attribute. </h3>
</div>
</body>
</html>
```

myscript.js

```
alert(" Hello World. \n Welcome to the javaTpoint.com\n
 This is an example of the defer attribute.");
```

JavaScript redirect

Redirect is nothing but a mechanism of sending search engines and users on a different URL from the original one. The redirected page can be on the same server or on a different server. It can also be on the same website or on different websites. Sometimes when we clicked on a URL, we directed to another URL. It happen because of the page redirection. It is different from refreshing a page.

```
<link rel ="canonical" href ="https://www.javatpoint.com/"/>
```

location.replace()

It is one of the commonly used window.location object. It is used for replacing the original document with a new one.

In this method, we can pass a new URL, and then it will perform an HTTP redirect. It is different from href as it removes the current document from the document's history, so it is not possible to navigate back to the original document.

```
Window.location.replace("new URL");
```

Example

```
<html>
<head>
<Sript type ="text/javascript">
function page_redirect(){
Window.location="https://www.javatpoint.com/";
}
</Script>
```

```
</head>
<body>
<h2>This is an example of the page redirection</h2>
<p> Click the following button to see the effect.</p>
<form>
<input type="button" value="Redirect" onclick =
        "page_redirect()"/>
</form>
</body>
</html>
```

JavaScript Scope

A scope can be defined as the region of the execution, a region where the expressions and values can be referenced.
There are two scopes in JavaScript that are globle and local:

Global Scope:
In the global scope, the variable can be accessed from any part of the JavaScript code.

Local scope:
In the local scope, the variable can be accessed within a function where it is declared.

Example

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
```

```
<script>
Var $var12=200;
function example() {
    Var $var12 = 300;
    document.Write("inside example() function = " + $ var12);
}
document.Write("Outside example() function =" + $ var12);
document.Write("<br>");
example();
</script>
</body>
</html>
```

JavaScript scroll

The onscroll event in JavaScript occurs when a scrollbar is used for an element. The event is fired when the user moves the scrollbar up or down. We can use the CSS overflow property for creating a scrollbar.

In HTML

We can use the onscroll attribute and assign a JavaScript function to it. We can also use the Java Script's addEventListener() method and pass a scroll event to it for greater flexibility.

In HTML

```
<element onscroll = "fun()">
```

In JavaScript

```
Object.onscroll = function() {myScript};
```

In JavaScript by using the addEventListener() method

```
Object.addEventListener("scroll", my script);
```

JavaScript Sleep/Wait

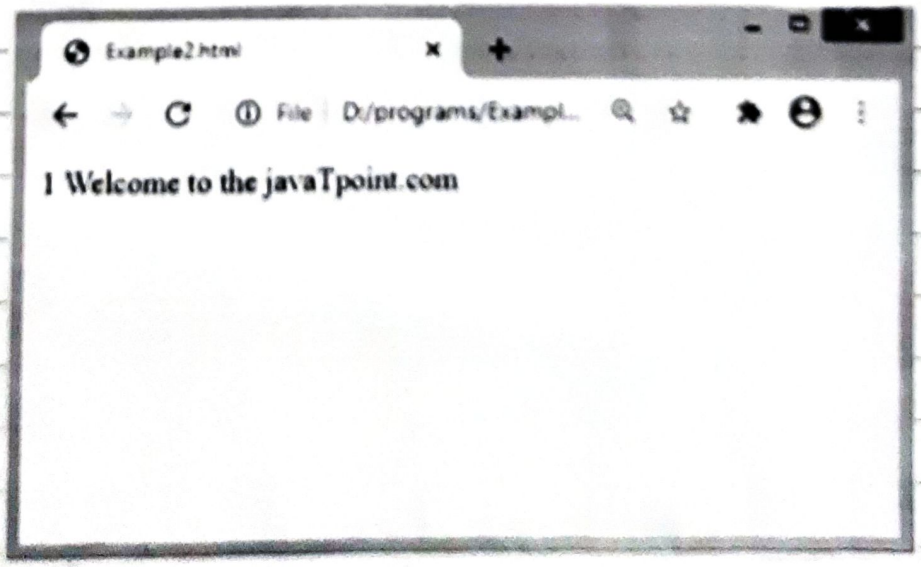
The programming languages such and PHP, C has a sleep(sec) function to pause the execution for a fixed amount of time. Java has a thread.sleep(), phyton has time.sleep(), and Go has time.Sleep (2*time.second)

Unlike other languages, JavaScript doesn't have any sleep() function. We can use some approaches for simulating the sleep() function in JavaScript. The features such as promises and async/await function in JavaScript helped us to use the sleep() function in an easier way. The await is used to wait for a promise and can only be used in an async function. The behavior of JavaScript is asynchronous, so there is a concept of promises to handle such asynchronous behavior of this asynchronous behavior. Its continues work and does not wait for anything during execution. Async/await functions help us to write the code in a synchronous manner.

Example

```
<html>
<head>
</head>
<body>
<h1>Example of using sleep() in JavaScript</h1>
<Script>
  function sleep (milliseconds){
    return new Promise(resolve => setTimeout(resolve,
                                milliseconds));
  }
  async function fun() {
```

```
document.write('Hello World');
for(let i = 1; i <= 10; i++) {
    aWait sleep(2000);
    document.write(i + " " + "Welcome to the javaTpoint.
                com" + " " + "</br>");
    }
}
funU;
</script>
</body>
</html>
```



JavaScript: void(0)

The Void operator is used to evaluate an expression and returns the undefined. Generally, this operator is used for obtaining the undefined primitive value. It is often used with hyperlinks. Usually the browser refreshes the page or loads a new page on clicking a link. The javascript: void(0) can be used when we don't want to refresh or load a new page in the browser on clicking a hyperlink.

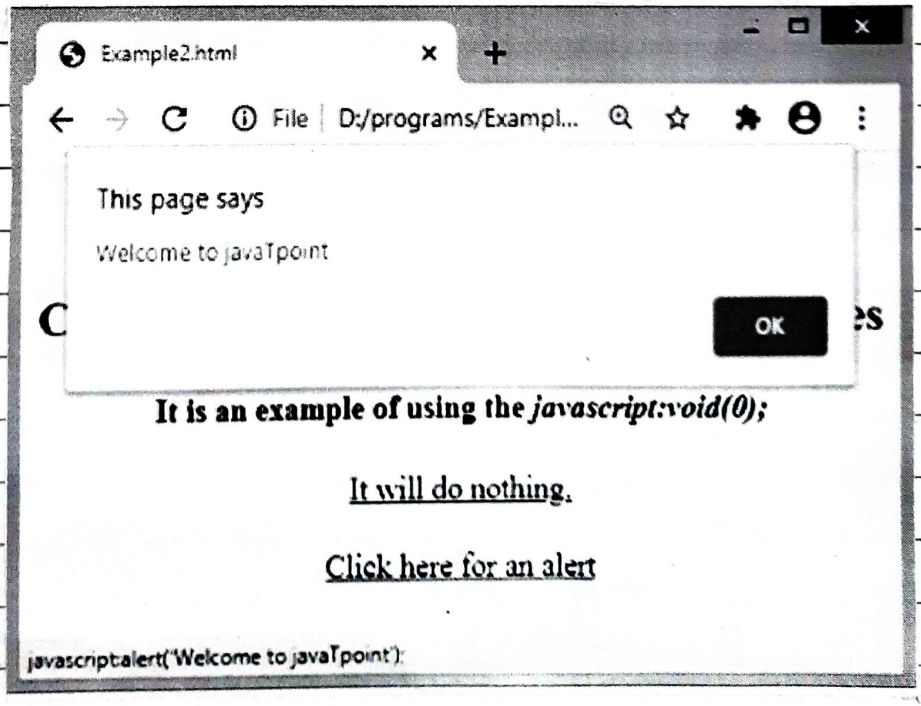
We can use the operand 0 in two ways that are void(0) or void 0. Both of the ways work the same. The JavaScript: void(0) tells the browser to "do nothing" i.e., prevents the browser from reloading or refreshing the page. It is useful when we insert links that have some important role on the webpage without any reloading. So, using void(0) on such links prevents the reloading of the page but allows to perform a useful function such as updating a value on the webpage.

It is also used to prevent unwanted redirecting of of the page.

Example

```
<html>
<head>
</head>
<body>
<center>
<h1> Hello World :):)</h1>
<h2> Click the following links to see the changes </h2>
<h4> It is an example of using the <i> javascript:
        void(0); </i> </h4>
```

```
<a href = "JavaScript:Void(0);">
It Will do nothing
</a>
<br/><br/>
<a href = "Javascript:alert('Welcome to the JavaTpoint');">
        Click here for an alert </a>
</center>
</body>
</html>
```



JavaScript Form

- Form name tag is used to define the name of the form. The name of the form here is "Login_form". This name will be referenced in the JavaScript form.
- The action tag defines the action, and the browser will take to tackle the form when it is submitted. Here, we have taken no action.
- The method to take action can be either post or get, which is used when the form is to be submitted to the server. Both types of methods have their own properties and rules.
- The input type tag defines the type of inputs we went to create in our form. Here we have used input type as 'text', which means we will input values as text in the textbox.
- Net, we have taken input type as 'password' and the input values as text in the textbox.
- Next, we have taken input type as 'button' where on clicking, we get the value of the form and get displayed.

Other than action and methods, there are the following usful methods also which are provided by the HTML Form Element

- Submit():
  The method is used to submit the form.

- reset():
  The method is used to reset the form values.

Referencing forms

Now, we have created the form element using HTML, but we also need to make its connectivity to

JavaScript. For this, we use the getElementById() method that references the html form element to the Java-script code.

```
let form = document.getElementById('subscribe')
```

Submitting the form

Next, we need to submit the form by submitting its value. for which we use the onSubmit() method. Generally to submit, we use a submit button that submits the value entered in the form.
The syntax of the submit() method is as follows:

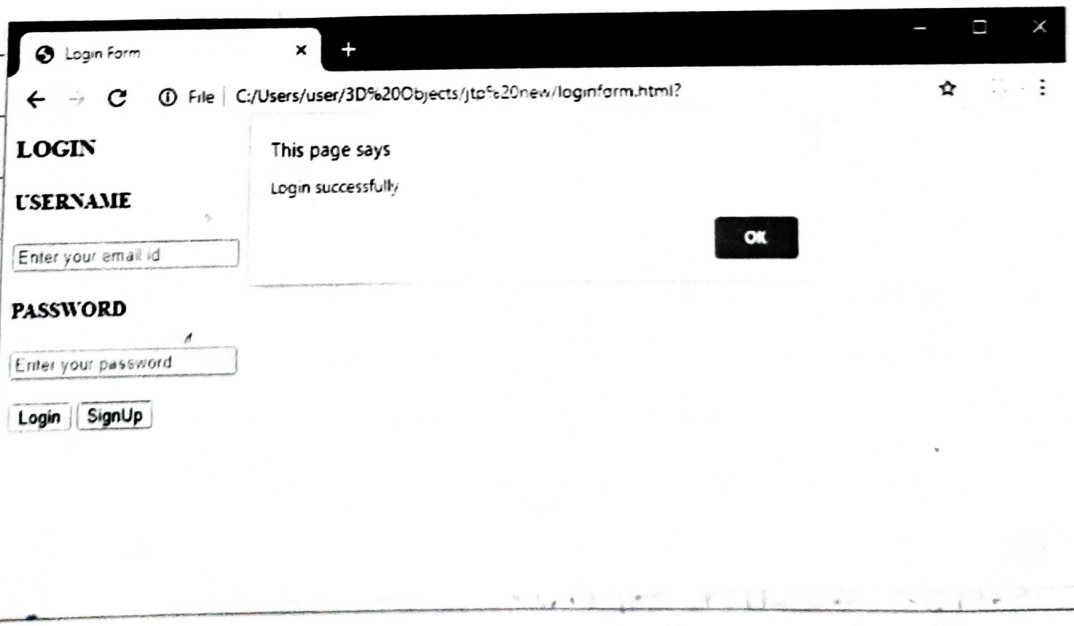
```
<input type = "submit" value = "subscribe">
```

When we submit the form, the action is taken just before the request is sent to the server. It allows us to add an event listner that enables us to place various validations on the form. Finally, the form gets ready with a combination of HTML and JavaScript code.
Let's collect and use all these to create a Login form and signup form and use both.

Login Form

```
<html>
<head>
<title>Login Form</title>
</head>
<body>
<h3> LOGIN </h3>
<formform="Login_form"Onsubmit="Submit_form()">
```

```
<h4> USERNAME </h4>
<input type = "text" placeholder = "Enter your email id"/>
<h4> PASSWORD </h4>
<input type = "password" placeholder = "Enter your password"
        /> </br></br>
<input type = "submit" value = "Login"/>
<input type = "button" value = "signup" onclick =
        "Create"/>
</form>
<script type = "text/javascript">
function submit_form(){
alert("Login Successfully");
}
function Create(){
window.location = "signup.html";
}
</script>
</body>
</html>
```



SignUp Form

```
<html>
<head>
<title> SignUp Page</title>
</head>
<body align = "center">
<h1> CREATE YOUR ACCOUNT </h1>
< table cellspacing = "2" align = "center" cellpadding = "8" border = "0">
<tr > <td> Name </td>
<td ><input type = "text" placeholder = "Enter your name" id =
        "n1"></td></tr>
<tr><td> Email </td>
<td><input type = "text" placeholder = "Enter your email id"
        id = "e1"></td></tr>
<tr><td> Set Password </td>
<td><input type = "password" placeholder = "Confirm your
        Password" id = "p2" > </td></tr>
<tr><td>
<input type = "Submit" value = "Create" onclick = Create_account
        ()"/>
</table>
<Script type = "text/javascript" >
function create_account() {
Var n = document. getElementById ("n1").value;
Var e = document. getElementById ("e1"). value;
Var p = document. getElementById ("p1") value;
Var cp = document. get ElementById ("P2") value;
// code for password validation
        var letters = /^(A-ZA-z)+$/;
        var email_val = /^([a-zA-Z0-9_|.|-])+|@(([a-zA-
            Z0-9\-])+\.)+ ([a-zA-Z0-9]{2,4})+$/;
// other validations required code
```

```
if (n==""||e==""||p==""||cp==") {
alert("Enter each details correctly");
}
else if (!letters.tests(n))
        {
         alert('Name is incorrect must contain alaphbets only');
        }
elseif(!emailval.test(e))
        {
         alert('Invalid email format please enter Valid email id');
        }
elseif(p!=cp)
{
alert("Passwords not matching");
}
else if(document.getElementById("p1").value length>12)
{
alert("Password maximum length is 12");
}
else if(document.getElementById("p1").value.length<6)
{
alert("Password minimum length is 6");
}
else{
alert("Your account has been created sucessfully...
        Redirecting to JavaTpoint.com");
window.location = "https://www.javatpoint.com/";
}
}

</Script>
</body>
</html>
```

Adding JavaScript to HTML Pages

There are following three ways in which users can add JavaScript to HTML Pages.

1. Embedding Code
2. Inline Code
3. External Code

1. Embedding code

Example

```
<!DOCTYPE html>
<html>
 <head>
  <title>page title</title>
  <Script>
  document.Write("Welcome to Javatpoint");
  </script>
  </head>
   <body>
    <p>In this example we saw how to add Javascript in the
              head section</p>
     </body>
     </html>
```

2. Inline Code

Example

```
<!DOCTYPE html>
<html>
<title> page title</title>
</head>
<body>
```

```
<p>
<a href="#" onclick="alert('Welcome!')">clickme</a>
</p>
<p> in this example we saw how to use inline JavaScript
    or directly in an HTML tag. </p>
</body>
</html>
```

3. External file

```
<html>
<head>
<meta Charset="utf-8">
<title>including a External JavaScript File</title>
</head>
<body>
<form>
<input type="button" value="Result" onclick="display()"/>
</form>
<Script stc="hello.js">
</script>
</body>
</html>
```

What is hoisting in JavaScript?

In JavaScript, Hoisting is a kind of default behavior in which all the declarations either variable declaration or function declaration are moved at the top of the scope just before executing the program's code. However, it can be considered an advantage because all funtions and variable declarations are placed to top of their scope no matter where they are all declared anywhere in the whole program.

even regardless of whether they are declared global or local.
Due to the concept of hoisting in JavaScript, we can call a function even before we define the function definition in our program's code.
In simple words, we can say that we can use the variables and functions in JavaScript before declaring them because as we disscussed above JavaScript compiler moves the declarations of all the variables and functions at the top of their scope so that there will not be an error of any kind. The concept of JavaScript of moving all declarations of the variable and functions to the top of their scop by compiler itself just before the execution of code is known as Hoisting.

What is the difference between Java and JavaScript

	Java Language	JavaScript Language
1.	It is a programming language.	It is a scripting language.
2.	Java is a pure Object Oriented programming Language.	Java Script is object-based Language.
3.	Java is a standalone language.	JavaScript is not a standalone language, as it needs to be integrated into an HTML program for execution.

4.	Java is a strongly typed language. Which means that the user has to decided the data type of the variable before declaring and using it. Example "int a", the variable "a" can store the value of integer type only.	JavaScript is a loosely typed language. Which means that the user does not have to worry about the data-type of the variable before and after the declaration. Example "var a", the "a" variable can store the value of any data-type.
5.	Java program should be compiled before execution.	JavaScript needs to be integrated into the HTML program for the execution.
6.	The Web-browser is not required to run java programs.	The Web-browser is essential to run the JavaScript programs.
7.	It is one of the complex languages to learn.	It is one of the easy languages to learn.
8.	In Java, by utilizing the multithreading, users can perform complicated tasks.	In JavaScript, user is not able to perform complicated tasks.
9.	It requires a large amount of memory.	It does not require that amount of memory.
10.	Java programming language was developed by the "Sun Microsystems."	JavaScript programming language was developed by the "Netscape".

11.	In Java programming language. programs are saved with the ".Java" exentsion.	On the other hand, programs in JavaScript are saved with the ".js" extension.
12	Java is stored on the host machine as the "Byte" Code.	JavaScript is stored on the Host machine (client machine) as the "source" text.